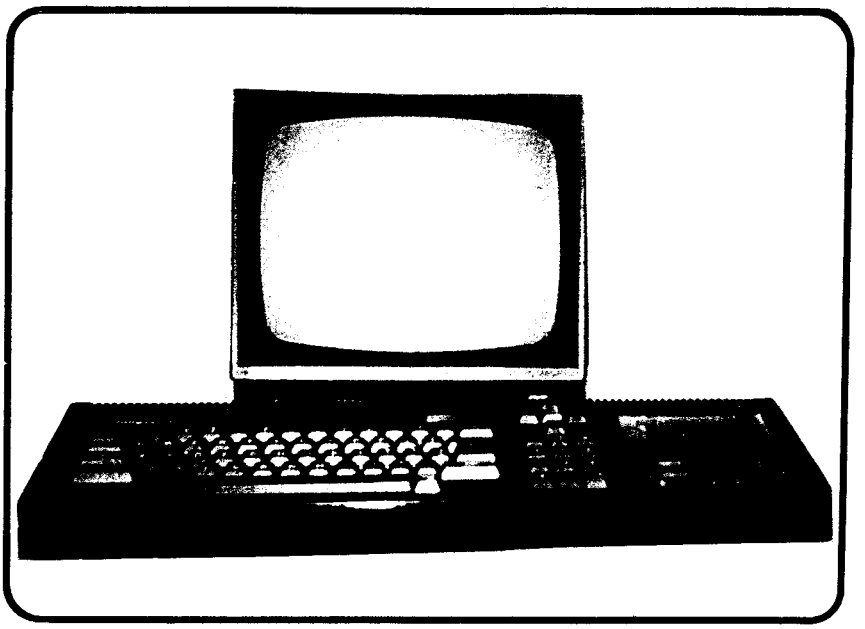
PRINT—OUT

Price 70p

ISSUE FIVE



By Thomas Defoe, Mark Gearing and Jonathan Haddock
Contributors - Bob Taylor, Richard Sergeant and Alan Scully

Including :

INDEX

Miscellaneous

Features

Reviews

Programming

We would like to express our thanks to Mr Gearing and BLACK HORSE AGENCIES, JANUARYS, for the continued use of their photocopier in the production of this Issue of Print-Out.

Editorial Issue 5

WELCOME TO ISSUE FIVE OF PRINT-OUT

Once again I am writing the editorial to another issue of Print-Out - it hardly seems as if 2 months can have passed. However, this is the largest issue to data and several new series have started which we hope that you will enjoy !!! We are still looking for more people to help with the magazine, especially with articles to do with BASIC programming. Again we would like to express our thanks to Bob Taylor, who has again contributed to this issue, and also to Alan Scully & Richard Sergeant who have both written articles for Issue Five.

Issue Six (is it really that many?!) will hopefully be completed by about the 30th July. If you wish to order a copy of this in advance, then please send one of the following :-

a) 70p plus an A4 SAE (with a 28p stamp)
b) £1.10 which includes postage & packing

For more details of this and other offers, please turn turn to the 'Offers' & 'Subscriptions' sections at the back of the magazine. For those of you who don't know, we advertise each new issue of the magazine in Amstrad Action's Small Ads section about once every two months.

Of course, if you have any questions or problems concerning the CPC then please do not hesitate to write to us at the address below and we'll do our best to solve it. The same address should also be used for any other magazine related enquiry.

The address to write to is :-
PRINT-OUT, 8 Maze Green Road,
Bishop's Stortford,
Herts CM23 2PJ.

Beginner's BASIC

We have now looked at about twenty BASIC commands and studied some of their uses. In this issue, I plan to introduce a few more commands & also consolidate some of the points we have already covered.

In a large program, there might be some parts which need to be used several times during its execution or, perhaps, a section of it which has to be accessed from different places. If we wanted to, we could write out these bits of the program over and over again whenever we wanted to use them. However, this would make the program much longer than necessary and possibly more complicated. In a simple program, such as below, you can see that several parts are the same.

```
 10 INPUT "What is your first number ";number1
 20 INPUT "What is your second number ";number2
 30 average=(number1+number2)/2
 40 PRINT "The average of your two numbers is";average
 50 total=average
 60 INPUT "What is your first number ";number1
 70 INPUT "What is your second number ";number2
 80 average=(number1+number2)/2
 90 PRINT "The average of your two numbers is";average
100 total=total+average
110 INPUT "What is your first number ";number1
120 INPUT "What is your second number ";number2
130 average=(number1+number2)/2
140 PRINT "The average of your two numbers is";average
150 total=total+average
160 PRINT "The total of your three averages is";total
```

It would save time and space if these lines only needed to be entered once. Fortunately, there's a way to do this and it uses an idea known as a SUBROUTINE and needs two new commands GOSUB and RETURN. Look at the lising below.

```
 10 GOSUB 90
 20 total=average
 30 GOSUB 90
 40 total=total+average
 50 GOSUB 90
 60 total=total+average
 70 PRINT "The total of your two averages is";total
 80 END
 90 INPUT "What is your first number ";number1
100 INPUT "What is your second number ";number2
110 average=(number1+number2)/2
120 PRINT "The average of your two numbers is";average
130 RETURN
```

Although this program is not very useful, it does illustrate the benefit of subroutines very well, and is probably one of the shortest programs to do this. The obvious advantage is that it's three lines shorter but there are other benefits which we will deal with later. First of all, we need to look at the three new commands that we've included, GOSUB, END and RETURN.

GOSUB occurs in lines 10,30,50. It stands for GOto SUBroutine and is always followed by a line number. When the computer encounters this command, it checks to see if the specified line number exists (and if not it tells you that it has a problem). If it finds the correct line number in the program, it goes to that line, missing out any of the program in between, and starts executing the program from its new position. However, at the same, it stores the line number from which it jumped for later reference. It then executes the instructions which it finds in the subroutine (in the above example the subroutine lasts from line 90 to line 130 and in line 10 the instruction GOSUB 90 tells the computer to go to line 90 ie. the start of the subroutine).

When it next encounters a RETURN command, it finds out where it last used a GOSUB command and then goes back to the line after the one which contained the actual GOSUB command. The small diagram below will hopefully make this clearer.

```
 10 GOSUB 90
 20 Next line of the program after GOSUB
 30 ...

... 80 END
 90 This is where the subroutine starts
100 Instructions in the subroutine that
110 are executed as normal until it has
120 a RETURN command.
130 RETURN
```

Line 80 also contains a new command, END. There are no prizes for guessing what this does; it ends the program. When the computer encounters END it stops the computer from executing any more commands from the program until it is RUN again. The reason for this is very important if we look at the operating order of the program.

1. Line 10 sends the computer to line 90.
2. It executes the instructions from line 90 to 130 until it meets a RETURN command at which point it jumps back to line 20 (executed normally).
3. When it executes line 30 it goes to line 90.
4. It now does the same as it did in part two except it now RETURNs to 40.
5. Line 50 tells the computer to GO to the SUBroutine at line 90
6. The computer now does part two again but returns to line 60.
7. Line 70 is executed normally.
8. Line 80 now ends the execution of the program in order to stop the computer accidentally doing lines 90 to 120 again.

When the computer gets to line 80, unless there was an END command it would just carry on executing lines 90 to 120 as if they were part of the normal program rather than a special subroutine. If the computer tried to run line 130 it would not be able to find a GOSUB command which had told it to go to line 90, & it would thus be unable to RETURN and would print up an error message.

With a program it is possible to have large numbers of subroutines that are either jumped to (using GOSUB) separately or indeed from within another subroutine. Look at the program below.

```
 10 INPUT "What is your name ";name$
 20 GOSUB 70
 30 INPUT "What is your friend's name ";name$
 40 GOSUB 110
 50 PRINT "I think I will like ";name$;
 60 END
 70 PRINT "That is a nice name ";
 80 GOSUB 110
 90 PRINT name$
100 RETURN
110 name$=UPPER$(name$)
120 RETURN
```

Again this is not a very exciting program but it illustrates another useful point to do with subroutines – the same subroutine can be used from a different part of the program and in a slightly different way. It also shows that you can have more than one subroutine in a program at a time and also reuse variables.

If you study the above program and follow its path of execution, you should be able to see that there is one subroutine from 70-100 and another from 110 to 120. The first subroutine is called from the main program, prints some text and then goes to the second subroutine. Now, when a RETURN is met in the 2nd subroutine, the program jumps back to line 90 in the first subroutine. It prints out a string (which was made into capital letters by line 110) and then returns to the main program. Line 40 executes the second suboutine directly without having to go throught the first subroutine and when the RETURN is encountered the computer goes back to line 50.

Although this may sound quite complicated, if you trace the program's path with your finger and obey the instructions (such as the GOSUBs and the RETURNs) you will see that it really does work. However, BASIC has an easy way of removing any confusion over subroutines – the REM command. In order to see it work, enter or alter these lines to the above program.

```
 20 GOSUB 70 : REM Print message and convert to uppercase
 40 GOSUB 110 : REM Convert string to uppercase
 80 GOSUB 110 : REM Convert string to uppercase
 69 REM Subroutine to print message and convert to uppercase
109 REM Subroutine to convert to uppercase
```

With REM the computer ignores everything after the REM and so this will have no effect at all on the way the program runs. See you in a couple of months time.

HOMEBREW SOFTWARE

Rebound – BY ALAN SCULLY

(Price – £2 on tape or £4 on disc)

This is an excellent game from Alan Scully who, with Rebound and his other progam 'One Arm Bandit Simulator' has proved himself to be a most professional games writer.

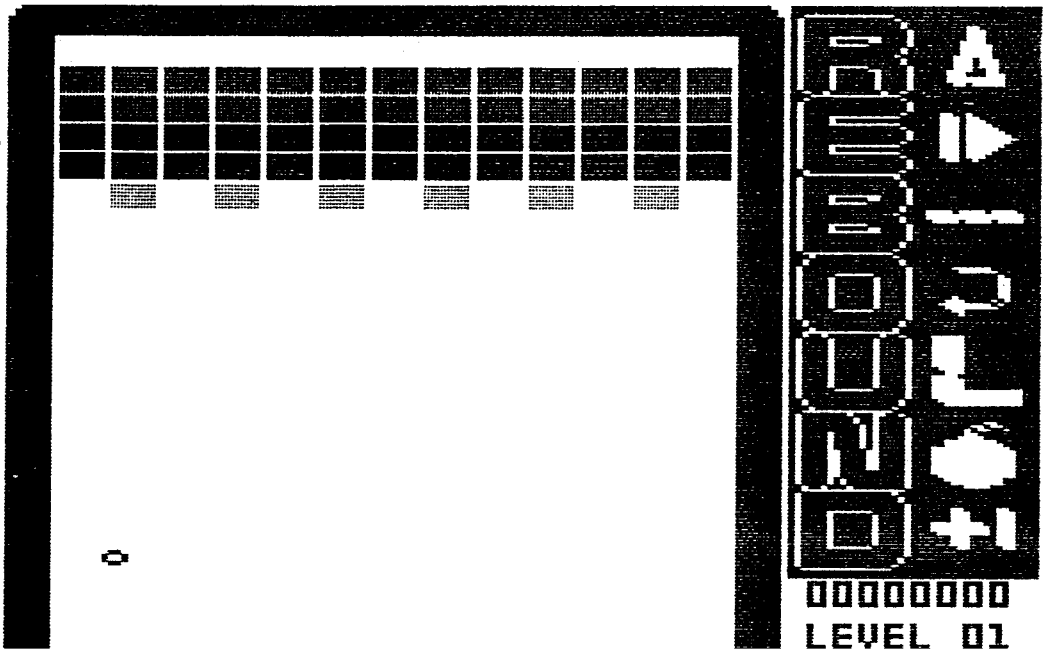
Rebound is a game of patience & skill requiring the player to knock a ball around the screen and it can bounce off the walls making bricks disappear when the ball hits them. The gameplay is simple and easy to understand with over 40 different screen layouts. The screens themselves are well laid out, neatly and professionally done. All the information needed is clear and uncluttered so at a glance the necessary facts can be easily seen.

The actual game screen is made up of bricks that together form some shape or pattern. The aim of the game is to hit all the bricks (some more than once) without loosing a life. Once they have all disappeared the player moves on to the next screen. A password is needed every ten screens. Unlike most games of this type there is no bat & a ball is lost when it passes through a small hole at the base of the screen. When the ball hits the base, either by bouncing to the ground or by being recalled by the player, it moves quickly towards one of the holes and it is upto the player to launch it in time.

The gameplay is made more interesting by the fact that behind some of the bricks lies a special block which, when collected, lights up an icon. When the desired icon is activated, it either slows down the speed of the ball, blocks up the exit, adds blocks to the screen or gives you a bonus life.

Rebound is so good that I almost forgot I was playing a homebrew game & it is a credit to Alan Scully that he has managed to produce a most professional piece of software.

Having said that, I found the game slightly tedious & frustrating at times, especially when just one brick was left on the screen – sometimes I was knocking the ball around the screen for at least ten minutes! Knowing the passwords solved this problem as I could get onto the more exciting & challenging screens. But with a price of only £2 (tape) & £4 (disc) who can fail to gain if they buy this game ?!! It is excellent in almost every respect !



One Arm Bandit Simulator

– BY ALAN SCULLY

(Price – £2 on tape or £4 on disc)

This is the second of Alan Scully's homebrew games. The gameplay this time does not involve balls and bricks but fruit machines and gambling!!!

Unlike Rebound, this game does not have a proper loading screen, but clear instructions do appear before the game is played. The screen layout itself is good with clear graphics but is perhaps slightly cluttered. It gives the impression that too much has been crammed into too small a space.

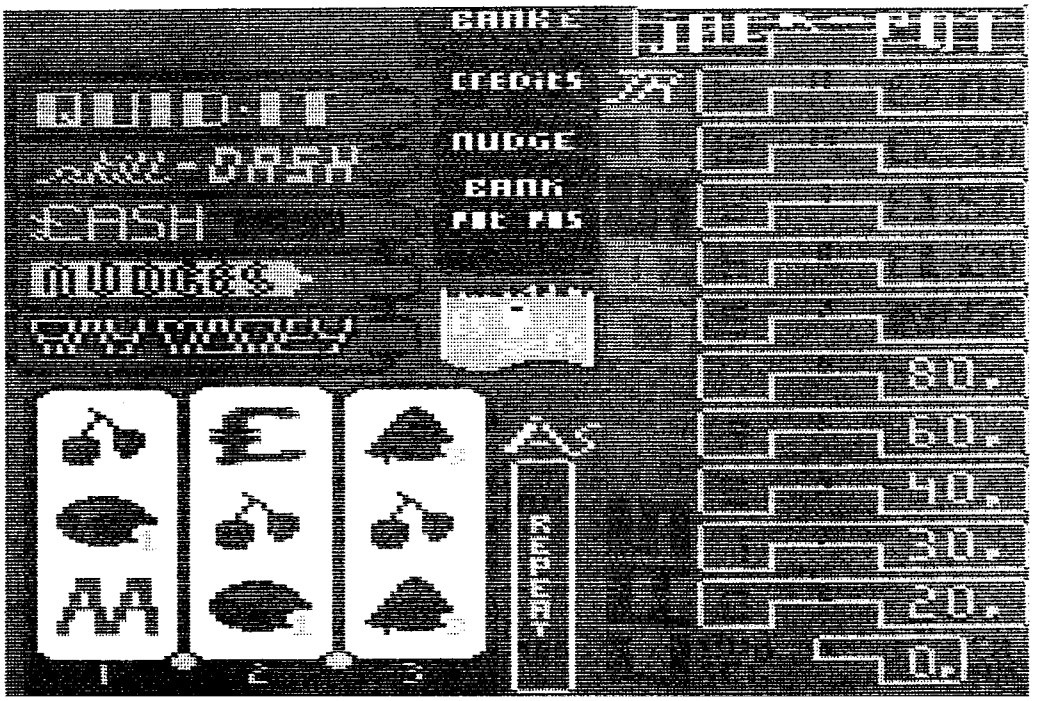
It took me a while to know exactly what was happening as the instructions, although clear, were concise and did not tell you everything – there is a certain amount of working out needed on the player's part. But the game is fun to play & it really does generate excitement when the player wins or loses money. There are a number of choices available to the player if he gets a high score on the fruit machine. For example he can win nudges (these allow the reels to move on one notch instead of spinning totally). The player can also play many special games such as 'CashDash', 'Any Money' and lots more if the right combination of symbols is gained.

Overall the game is a good one, not quite as excellent as Rebound, but then that is a tough game to beat. This game, I thought, had more staying power than Rebound and, for me, it was more fun to play but it depends upon your personal preference. If you are the gambling type, then this is surely the game for you, if not then it is worth buying as it is only £2 on tape, £4 on disc. However if you buy both games then together they will cost only £3 on tape or £5 on disc. This is an incredible bargain and if you only get half as much enjoyment out of the games as I did, you will be very satisfied.

To sum up, it's not quite as outstanding as Rebound but is still very good.

'REBOUND' and 'ONE ARM BANDIT SIMULATOR' both cost £2 on tape or £4 on disc but if you buy them together as a pair, they cost a mere £3 on tape and £5 on disc. They can both be obtained from :-

Alan Scully, 119 Laurel Drive, Greenhills,
East Kilbride, Glasgow G75 9JG.



PART 5:
ENTERING & STORING

Machine Code

As I have mentioned before, one of the best ways of learning how to program in any computer language is to set yourself a task and then to try and design a routine which performs this.

The brief for this issue is to 'write a program that prints an introductory message, asks a question of the user, gets the answer and then checks to see if the answer is correct. If it is, a message of congratulations is printed otherwise the right answer is displayed. To make this more complicated, the question will require a word as the answer rather than a number.

STRUCTURED PROGRAMMING

The best method of writing programs is known as STRUCTURED PROGRAMMING and this involves logical and easy to follow steps. They are often made up of small modules which can be tested and used in isolation from the rest of the program. The program that we have to write contains several major parts as shown below:-

1. PRINT the question
2. INPUT the answer
3. CHECK the answer
4. PRINT the outcome

You will notice that parts 1 and 4 both involve printing and so we can use the same routine for them. As we have already written a subroutine that we can use for this purpose, we don't need to write another one. For those of you who are new to Print-Out it is listed here but we looked at it in more detail in Issue One on page 33.

```
          ORG &4000              ; locate the program at address &4000
          LD HL,string           ; load HL with the address of the text
          CALL print             ; call the subroutine called '.print'
          RET                    ; return to BASIC

.print    LD A,(HL)              ; A = the contents of the address (HL)
          CP 0                   ; see if A holds zero (FLAGS = A-0)
          RET Z                  ; return if the A holds zero (ie ZERO=1)
          CALL &BB5A             ; print the character in A
          INC HL                 ; HL=HL+1
          JP PRINT               ; jump to the label print

.string   DB "Printing",0        ; the text to print
```

However, we need to print different pieces of text and to do this, all we need to do is to load HL with the address at which the text is stored and then call the subroutine '.print'.

INPUTTING

Next, we must write a module which will allow us to input a word, number or sentence. However, not only must we input a word and print it to the screen but we must also store it in memory so that it can be used again later in the checking routine. Such a module will be much more complex than anything we've looked at before and so it may be helpful to jot down a checklist of all the things we need to include :-

1. We need to actually get the letter from the user.
2. The routine must detect when inputting has been ended.
3. The DELETE key also needs to be detected.
4. We need to let the user see what has been entered.
5. There must also be a maximum length imposed on the word.
6. The end of the store needs to be identified.

We have now got to work out how we are actually going to achieve these things & to do this we will take each point in turn and try and come up with a solution.

The first thing that we need to do is to get the letter that the user pressed & to do this we will use the firmware call, KM_WAIT_CHAR (&BB06), which we looked at in Issue Three. Basically, KM_WAIT_CHAR waits until the user presses a key & then stores the ASCII code of the character that was pressed in the A register. For example if the 'L' key was pressed then the A register would hold the value of 76 (or &4C).

In order to terminate inputting, all we do is to compare the ASCII value in the A register with 13 (the code for ENTER) and if it has been pressed, jump to the next part of the program.

Next, we need to check and see if the DELETE key has been pressed. The way that we will do this is to see whether the A register holds the number 127 (which is the ASCII code for DELETE) after we have called KM_WAIT_CHAR. If it does, we'll need the program to jump to a special routine to handle erasing a letter. This routine has to do two things - it must remove the character from the screen and also erase the letter from the store in memory. We must also be careful not to allow the computer to accidentally erase any of the memory that is not part of the store. This will effectively clear some of the program and this might cause a crash. In order to overcome this, a counter is needed to record the number of characters that have been entered.

If neither ENTER nor DELETE have been pressed we must let the user see that the letter they entered has been accepted and also remind them what they have typed so far. To do this we'll use the firmware call TXT_OUTPUT (&BB5A) which we have used in every issue so far and prints the character that is contained in the A register (by means of its ASCII code).

To make sure that a word that would overwrite something vital in memory is not inputted, we will impose a maximum length on the word that can be entered & use a counter to make sure that this limit is not exceeded.

Finally, we need to tell the computer where the stored word ends in its memory so that the checking routine can work properly. To do this we'll add a byte of value 0 to the end of the word.

THE LISTING

Printed here is a short program that will do what we require and it is made of two subroutines which call each other. The main subroutine, labelled .input, does the actual inputting and when it encounters the [DELETE] key it then calls the subroutine, .erase. When inputting is finished, the computer returns to the main program. For this to work, HL must hold the address of the place where the inputted statement is to be stored and B, which is the counter, needs to hold 0 before .input is called. When inputting has finished, the instruction LD (HL),0 adds a byte of 0 onto the end of the store to indicate the place where it ends.

```
        ORG &4000       ; program located at &4000
        LD HL,store     ; HL holds the address of the label '.store'
        LD B,0          ; B equals 0
        CALL input      ; goto the subroutine '.input'
        LD (HL),0       ; the contents of the address in HL equal 0
        RET             ; return to BASIC

.input  CALL &BB06      ; get a character, store its ASCII code in A
        CP 13           ; see if A equals 13 (FLAGS = result of A-13)
        RET Z           ; return if the zero flag is set (ie. A equals 13)
        CP 127          ; see if A equals 127 (FLAGS = result of A-127)
        JP Z,erase      ; jump to .erase if zero is set (ie. A equals 127)
        CALL &BB5A      ; print a character according the value in A
        LD (HL),A       ; the contents of the address in HL equal value of A
        INC HL          ; HL=HL+1
        INC B           ; B=B+1
        LD A,B          ; A=B (A now holds the number of letters entered)
        CP 15           ; see if A equals 15 (FLAGS = result of A-15)
        RET Z           ; return if zero flag is set (ie. A equals 15 which
                        ; means that 15 characters have been entered)
.dummy  JP input        ; jump to the label '.input'
.erase  LD A,B          ; A=B (A now holds the number of letters entered)
        CP 0            ; see if A equals 0 (FLAGS = result of A-0)
        JP Z,dummy      ; jump to '.dummy' if zero flag is set (ie. A equals
                        ; 0 which means that no characters have been entered)
        DEC B           ; B=B-1
        DEC HL          ; HL=HL-1
        LD (HL),0       ; the contents of the address pointed to by HL equal 0
        LD A,8          ; A=8 (ASCII code for backspace)
        CALL &BB5A      : a backspace is printed to move cursor back one
        LD A,32         ; A=32 (ASCII code for space)
        CALL &BB5A      : a SPACE is printed to erase previous character
        LD A,8          ; A=8 (ASCII code for backspace)
        CALL &BB5A      ; a backspace is printed to move cursor back one
        JP dummy        ; jump to the label '.dummy'
.store                  ; label to show where the inputted word is stored
```

HOW IT WORKS....

The main body of the program sets up the register values that are required for the program to work and then the input subroutine is called. First of all, a character is inputted using CALL &BB06 and the next instruction tests to see if it is character 13 (RETURN). You might remember, from last issue, that what actually happens is that 13 is subtracted from the A register & then the flags are set accordingly although the actual result of the calculation isn't stored anywhere; this is what the strange comment 'FLAGS = result of A-13' represents. However, if the character that was inputted did equal 13, the ZERO FLAG would be set and the instruction RET Z would return to the main program. If the value of A was not 13, then the ZERO FLAG would not be set and the computer would not RETURN.

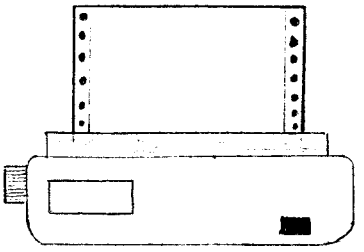
The DELETE key has a value of 127 and the next 2 lines check to see whether it has been pressed. If it has the program jumps to the routine labelled .erase but otherwise it carries on as normal. The character is then printed using CALL &BB5A, the address pointed to by HL has the value of A placed in it and HL (the address) is increased by one as is B (the counter) which tells us the number of letters that have been entered. Then the program checks to see if B is equal to 15 (ie. 15 characters have been entered) and if so, it terminates the inputting routine otherwise this process is repeated. The .store label indicates where an inputted word should be placed in memory (ie. after the program).

ERASING

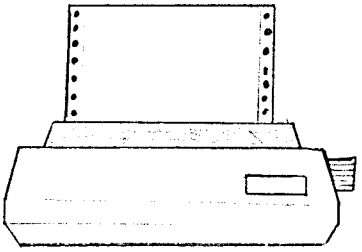
Upon entering the routine, HL holds the address where the next character is to be stored and B holds the number of characters that have been entered so far. Therefore the first thing that the routine has to do is to make sure that it is not deleting something that isn't part of the stored word. The way it does this is to see if B equals zero. If it does this means that no letters are present & so nothing can be deleted. It then jumps to .dummy & bypasses the print routine. However, if B does not equal zero it means that there is something which can be deleted and so it allows the delete routine to continue.

Unfortunately, B cannot be tested using the CP instruction and so the value of B must first be put into A. If there is a character to erase it decreases HL so that it points at the offending character & decreases B (ie the counter says that there is one less letter stored). It removes the letter from the store and then erases it from the screen by printing spaces and backspaces. It then jumps to .dummy in order to avoid the printing routine.

THIS ARTICLE IS CONTINUED ON PAGE 37. THE SECOND SECTION DETAILS HOW TO COMPARE TWO WORDS AND ACT ON THE DECISIONS. IT ALSO INCLUDES A COMPLETE LISTING OF THE PROGRAM ON HOW THE TWO PARTS FIT TOGETHER CORRECTLY.



~ PRINTER PROBLEMS



From the number of letters that we receive, the one item of hardware which causes most difficulties and problems is the printer. Therefore, we decided to write this article which will hopefully help to answer some of the more common problems and queries.

First of all, I'll deal with the question that crops up over and over again - What printer should I buy? This is also the hardest query to give a reply to, for the decision depends on what the individual wants to do, and how much money they have available. If money is no object then the simple answer every time is 'a Laserprinter' as it will give the cleanest and sharpest text and graphics of any printer. However, the price starts at about £1500 and rises steadily until it reaches between three and four thousand pounds. Most people do not have that kind of money to spend on a printer (especially one for a humble CPC which just has not got the power to drive a Laserprinter to the best of its capabilities - for that you really need a Macintosh with very complex programs and a couple of 30 megabyte hard discs!!!) The majority of potential printer owners are looking at spending between £150 and £700 at most so we will look at the printers which fall in this range.

There are three main types of printer and I'll make some generalised points about them first.

DOT MATRIX PRINTERS - These produce the letters/graphics by firing a series of pins in the correct pattern so as to form the desired character. Consequently the letters are fairly dotty if looked at closely. Because the image is made by punching dots onto the paper, graphics can be produced easily providing the correct screen dump is used & many have appeared in the national magazines over the years. There are now two types of dot matrix printer; the 9 pin and the 24 pin. At present, the 9 pin version is more common although many 24 pin printers are becoming popular. As you might expect the second type has 24 pins which can be 'fired' at the paper and, as they have more dots, the characters become more solid and less dotty. The drawback with these, is that the pins are arranged in a different shape (3 x 8 pins) from 9 pin printers (1 x 9 pins). Therefore, all the commercial graphic screen dumps for the CPC, which were originally written for just 9 pin printers, whilst working with the 24 pin variants produce rather 'blotchy' graphics - if screen shots, etc. are important to you and you are not able to write machine code screen dumps, I would think carefully about buying a 24 pin printer.

DAISY WHEEL PRINTERS, on the other hand, work in much the same manner as an electric typewriter (ie they have a disc which rotates until the correct letter is selected and then a hammer punches it onto the ribbon which in turn imprints it on the paper). This gives a very much improved type quality but it is slower and noisier than a dot matrix printer & is unable to print graphics. The other

disadvantage of daisywheel printers is that in order to print in different type styles, you need to buy new printing wheels (about £10 each) whereas dot matrix printers are supplied with with several typefaces built-in and more can easily be designed if needed.

The third type of printer is a LASERPRINTER which generally can print clean, shaded graphics and has excellent letter quality. They are also quiet, fast and efficient and can have new typestyles added very easily. However, they are not popular among home computer owners as most of their options are unavailable on the CPC. A fairly recent addition to the range of printers are INK JET PRINTERS. They are similar to laserprinters except that instead of using powder on light sensitive plates, the letters/pictures are formed by firing fine jets of quick-drying ink at the paper. Many of the advantages of a laserprinter are available but the price is about half that of a laser printer. As yet I have not heard of one being used on a CPC but it might be possible.

As a summary of the types of printer available, here is a chart showing their relative strengths and weaknesses

	DOT MATRIX 9-PIN	DOT MATRIX 24-PIN	DAISY WHEEL	LASERPRINTER	INK JET
SPEED	200cps (50cps)*	240cps (70cps)*	50cps	12 pages per minute	200cps
GRAPHCS	YES	YES	NO	YES	YES
ADVANTAGES	Text and graphics can be done;cheap and flexible;they have good support in many magazines		Produces very good text and is excellent for producing documents	Has excellent text/graphics and is fast & has many good builtin fonts	Has very good text/graphics & it is quite fast
PROBLEMS	'Dotty' letters & graphics on the 9 pin. 24 pin can't print clear graphics without some special program & has far improved text quality		Quite expensive;is unable to print graphics at all. Does not conform to Epson standards	Extremely expensive & has got virtually no support on the CPC;might not be at all compatable	Expensive and has virtually no support on the CPC;might not be at all compatable
PRICE	about £150 - £300		£400 - £700	£1500 - £3500	£700 - £1500

* Dot matrix have two speeds - one (draft) is very dotty and the other (Near Letter Quality) is the less dotty version. The number in the brackets refers to its NLQ speed and is given in characters per second (cps).

My general advice would be to buy a dot matrix printer and if graphics are not that important, preferably a 24 pin printer. If you feel that you are likely to print graphics of any kind then go for a 9 pin dot matrix printer and if colour is needed then the Star LC-10 Colour represents excellent value for money & you can judge its near letter quality type as the entire magazine is printed in it.

Also when considering buying a printer, remember that if you buy it by mail order, you are likely to receive a substantial discount. I've found that M.J.C. (see their advert in Amstrad Action) offer tremendous service & support and are always willing to give help and advice. Their telephone number is (0462) 32897/ 420847/421415 and their prices are often upto £100 lower than the retail price.

Error?!!

You should try & make sure that your printer is EPSON compatable (most are) and as this is the main industry standard you are guaranteed that most programs will work with your chosen printer. To connect your printer to your CPC you'll also need a CENTRONICS/PARALLEL cable and this leads us onto a common problem - the notorious double line feed.

If you find that the printer leaves a blank line between two lines of text. then you need to insulate pin 14 of the interface cable. The way to do this is to stick a small piece of tape over pin 14 of the CPC's printer port (pin 14 is the fourth in from the left on the top as viewed from behind) before the cable is plugged in.

Another common problem is the appearance of £ signs in place of # signs and vice versa. The problem is quite tricky to solve through programming but if you buy a good word processor you should be able to overcome this difficulty.

But perhaps the greatest problem of all is understanding the manual - they are the hardest books to learn from! When you have finally bought your printer, set it up, plugged it in, typed in the example listing and....the computer will probably reply with 'Syntax Error'. Sometimes the example will use the command LPRINT and you will need to replace this with PRINT #8. For example, if it says LPRINT "The printer works" you will need to enter PRINT #8,"The printer works". While this is fine for trying out the printer, there will come a time when you will want to produce documents/letters in an easier manner and then you need a word processor - I would recommend PROTEXT because of its availability on tape, disc and ROM.

When you first start using printers, you'll encounter many further problems but most of them can be sorted out through trial and error and careful reading of the manual. The great bonus with printers is that, short of taking a soldering iron to them, you are very unlikely to damage either printer or computer by experimentation - any messes you get yourself into can be overcome by switching the printer off for a couple of seconds.

If you have any further problems than please write and we would be only too glad to try and sort them out.

BITS AND PIECES

For tape owners, one of the greatest problems with the CPC is the time that it takes to load programs or files of all descriptions. I find with my 464 that there are many occassions when I wish to save the BASIC program that I am working on, load in another file (eg a program that I wish to check before MERGING) and then retrieve my old BASIC program, for example. Another time this happens, is when you need to use a utility at some point whilst programming. Of course, for disc owners this represents no real problem as disc accessing is so quick & simple. However, even then, there may be times when you wish you could make an instantaneous save of a program and then reload it just as quickly. This would allow you to make a backup copy of the program you are working on before making a major alteration, or imagine the uses of having two programs in memory at the same time and being able to swap between them easily. With these possibilities in mind, I programmed the short Machine Code routine below which stores a BASIC program at a safe address in memory for recalling later. The program can handle all BASIC programs which are less than 8K in size (it checks automatically and informs you if copying is not possible) although with a few minor modifications you could easily change the maximum size. The one thing it doesn't allow you to do is to halt a BASIC program in mid-execution, store it and then restart from that position as this would have required much more code to preserve variables, positions, etc.

The actual program sets up two RSXs, |STORE and |RECALL, whose code is held at &BE80 - because this area can survive a soft reset (CTRL-SHIFT-ESC) you need only to load the program once when you first switch on unless the code is overwritten by another M/C routine. The program that you wish to preserve is stored from &8000 to &A000 and this means that you have only 31K to use for your BASIC programs !!! Any stored program will survive a NEW command because of the line, MEMORY &7FFF, that prevents BASIC from clearing or overwriting any memory above this address. In order to run the program to run on a 664/6128 you will need to alter these two lines...

```
[F3]  30 DATA 95,BE,C3,B9,BE,01,66,AE
[9B]  80 DATA C9,21,17,BF,01,66,AE,7E
```

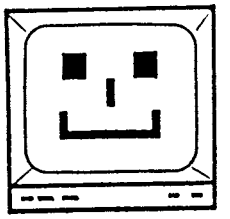
...it runs as printed on a 464. To use the RSXs, simply load in the program you wish to store, do whatever you want to with it (including running and editing it) and then type |STORE. If no error message has occured then all is well. Now load in another program or change the existing program and when you wish to retrieve your old, stored BASIC program type |RECALL. You will then be able to list, run, edit, etc the program as if it had been there all along. Please note that the stored program does NOT itself survive a soft reset but only the code.

```
[8F]  10 DATA 01,8D,BE,21,89,BE,C3,D1
[16]  20 DATA BC,00,00,00,00,0C,BF,C3
[D5]  30 DATA 95,BE,C3,B9,BE,01,83,AE
[9D]  40 DATA 21,17,BF,0A,5F,77,03,23
[42]  50 DATA 0A,57,77,62,6B,42,4B,11
[79]  60 DATA 00,20,A7,ED,52,D2,D2,BE
[15]  70 DATA 21,6F,01,11,00,80,ED,B0
[80]  80 DATA C9,21,17,BF,01,83,AE,7E
[A2]  90 DATA 5F,02,23,03,7E,57,02,21
[D8] 100 DATA 00,80,42,4B,11,6F,01,ED
[C7] 110 DATA B0,C9,21,E0,BE,7E,FE,00
[34] 120 DATA C8,CD,5A,BB,23,C3,D5,BE
[13] 130 DATA 54,68,65,20,42,41,53,49
[9D] 140 DATA 43,20,70,72,6F,67,72,61
[6C] 150 DATA 6D,20,69,73,20,74,6F,6F
[5B] 160 DATA 20,62,69,67,20,74,6F,20
[C8] 170 DATA 62,65,20,73,74,6F,72,65
[25] 180 DATA 64,0D,0A,00,53,54,4F,52
[B0] 190 DATA C5,52,45,43,41,4C,CC,00
[2C] 200 DATA end
[D6] 210 add=&BE80
[8F] 220 READ a$:IF a$="end" THEN GOTO 250
[53] 230 POKE add,VAL("&"+a$):add=add+1
[05] 240 GOTO 220
[01] 250 MEMORY &7FFF:CALL &BE80
[6C] 260 PRINT "Both RSXs now installed at &BE80"
[6B] 270 PRINT " To preserve a program type |STORE"
[6F] 280 PRINT " and to retrieve it use |RECALL"
[8E] 290 PRINT "NB Memory is now reduced to &7FFF"
```

Linechecker

A PROGRAM TYPING AID

All programs in Print-Out have Linecheck codes which are enclosed in brackets at the start of a line. Don't enter them in as they're designed to be used with Linechecker to eliminate errors when typing in programs which appear in this magazine. Please note, all programs will run whether Linechecker is being used or not. For information on how to use Linechecker, please see Issue Three.



The next program allows you to list any ROMs that you might have installed at any time (whether initialised or not). The main part of the program is written in Machine Code although the printing routine is driven from BASIC. Sometimes, when writing programs in either BASIC or Machine Code it is important to know which ROMs are available (eg. to check for clashes in commands names or memory that has been reserved). However, most commercial ROMs contain a |HELP command that lists the names of all the available ROMs. The major drawback is that you cannot use the information that the command found in your own programs as it's not stored anywhere (except on the screen). With the routine below the information that it finds out about the ROMs is stored in memory starting from &8045 and continuing until it has stored all the details of upto fifteen ROMs.
The order of storage is as follows :-

First byte	-	ROM type (0=foreground, 1=background, 2=extension
Second byte	-	ROM mark number
Third byte	-	ROM version number

Fourth byte - ROM modification number

Fifth byte onwards - ROM's name (end is signified by a byte of &FF)

The only problem with the listing is if the ROM is laid out in an unorthodox manner then the name may be corrupted (we have yet to find a ROM that does not work successfully and it has been tested with most of Arnor's ROMs and CPM).

```
[5B] 10 MODE 2:lin=2
[78] 20 DATA 11,45,80,0E,00,CD,0F,B9
[EF] 30 DATA C5,0E,00,C5,CD,0F,B9,CD
[E9] 40 DATA 1F,80,C1,0C,79,FE,0F,C2
[BA] 50 DATA 0B,80,C1,CD,18,B9,C9,21
[12] 60 DATA 00,C0,06,04,7E,FE,80,C8
[D9] 70 DATA 12,23,13,10,F7,4E,23,46
[7A] 80 DATA 0A,FE,80,F2,3C,80,12,03
[A2] 90 DATA 13,C3,30,80,D6,80,12,13
[5C] 100 DATA 3E,FF,12,13,C9,00,00,00
[2D] 110 DATA end
[8C] 120 add=&8000
[6C] 130 READ a$:IF a$="end" THEN 160
[54] 140 POKE add,VAL("&"+a$):add=add+1
[07] 150 GOTO 130
[08] 160 PRINT "The ROMs that are available are :-"
[6C] 170 CALL &8000
[09] 180 add=&8045
[46] 190 FOR rom=1 TO 15
[BF] 200 type=PEEK(add):mark=PEEK(add+1)
[39] 210 vers=PEEK(add+2):m=PEEK(add+3)
[D7] 220 add=add+4:name$=""
[2F] 230 c=PEEK(add)
[5A] 240 IF c=&FF THEN GOTO 270
[A4] 250 name$=name$+CHR$(c)
[CB] 260 add=add+1:GOTO 230
[6F] 270 add=add+1
[6E] 280 LOCATE 3,lin:PRINT name$
[20] 290 LOCATE 14,lin:PRINT USING "#.";
         mark;
[62] 300 PRINT USING "#";vers;:PRINT USI
         NG "#";m
[F3] 310 IF type=0 THEN t$="FOREGROUND":
         GOTO 330
[7E] 320 IF type=1 THEN t$="BACKGROUND"
         ELSE t$="EXTENSION"
[A1] 330 LOCATE 19,lin:PRINT t$;" ROM"
[4A] 340 IF PEEK(add)=0 AND PEEK(add+1)=
         0 AND PEEK(add+2)=0 THEN END
[F9] 350 lin=lin+1:NEXT rom
```

Continuing on this theme of listing any peripherals which your computer has attached, we have printed below a short routine that detects a disc drive (if attached and switched on).

```
      10 ON ERROR GOTO 1000
      20 disc=1:|DISC
      30 ' Rest of the program
     999 END
    1000 disc=0:resume 30
```

After running, disc will equal 1 if a disc drive was detected, and zero if a disc drive was not present.

An intro to R.S.X.s

part 1

by BOB TAYLOR

In common with other more advanced BASICs, Locomotive, the dialect resident in our CPCs, has the ability to have extra commands added. The name given to these is Resident System Extensions, or RSXs for short. In these few articles I hope to show the normal way of using this facility, together with my own rather unorthodox approach.

An RSX is a form of named CALL command: the latter has only an address to identify it while an RSX must have a name which is descriptive of its operation. Additionally, it is quite easy to type in a wrong address for a CALL with probably disastrous results; an RSX on the other hand will either go to the correct routine associated with the RSX name, or stop with an error message if the name has been entered wrongly. Of course both CALLed and RSX routines are present as machine code.

CAPS LOCK SETTER

As a starting point, we will look at the following routine which sets up 2 new commands for clearing or setting CAPS LOCK - intended for use within a program to set input to upper or lower case as required:

```
        ORG   &8000
.intrsx LD    BC,comtbl ;address of the command table.
        LD    HL,datblk ;address of the 4 byte data block.
        JP    &BCD1     ;return directly to BASIC after Firmware Call KL LOG EXT

.datblk DEFS 4          ;reserve 4 bytes for data block.
.comtbl DEFW namtbl     ;address of the name table.
        JP    capon     ;to routine to set CAPS LOCK.
        JP    capoff    ;to routine to clear CAPS LOCK.

.capon  LD    A,&FF     ;any non-zero value will set CAPS LOCK to on.
        JR    capcom    ;the rest of the routine is common with most of CAPSOFF.
.capoff LD    A,0       ;zero will reset CAPS LOCK (ie to lower case).
.capcom LD    HL,&B4E8  ;(&B632 for the 6128): the system variable holding the
                        ;CAPS LOCK flag.
        LD    (HL),A    ;store the new flag value.
        RET             ;to BASIC.

.namtbl DEFB "CAPSO","N"+&80
        DEFB "CAPSOF","F"+&80
        DEFB &00        ;end of Name Table byte.
```

WHAT IT DOES

In order to get the computer to recognise our two new commands, we need to present them to the Operating System via KL LOG EXT. The routine shown in the three lines starting at .intrsx is the shortest possible to achieve this. Note that instead of using 'CALL &BCD1:RET', we have JumPed to KL LOG EXT and so are using the RET at the end of that to RETurn from our initialisation routine also. This will save us one byte of space and 17 'T' states - reduction of the length of routines and their speeding up are always especial aims of my coding.

When KL LOG EXT is invoked, the Operating System logs on the Command Table. It does this by utilising the Data Block pointed to by HL. This is an area of 4 bytes which has to be in the central 32K of RAM. These bytes will be set up to contain the Command Table address together with the address of the next similar Data Block in a chain which ties together all separately entered RSXs and ROMs. If our RSX is the last in the chain, this address will be replaced with 2 bytes of &00 to signal that it's the last. These four bytes must not then be used for anything else. However, before the log-on, they don't have to be empty & unlike above, I often use part of the initialisation routine which becomes redundant once used, so saving four bytes of memory.

The Command Table (at .comtbl) which must also be in the central 32K of RAM, consists of two bytes which contain the address of the list of Command Names we are using for our RSXs. These two bytes are then usually followed by a series of JP (Jump) instructions to the routines which perform the RSXs (note that the sequence must be the same as the names in the Command Name table):

```
.comtbl DEFW address of Command Name Table
        JP   address of routine for RSX with FIRST name in Command Name Table
        JP   address of routine for RSX with SECOND name in Command Name Table
       (JP   address of routine for RSX with THIRD name in Command Name Table)
          etc.
```

I said usually, because we don't have to lay out each RSX's entry in the table in this way. If there is only one RSX being initialised, we can start the routine for the RSX immediately after the Command Name Table address and so save 3 bytes of memory. With just a few RSXs, we might be able to use JR instructions instead of JPs providing the routines are within 127 bytes of the entry. There is a necessity to maintain an area of 3 bytes for each entry (except for the last), so using JRs will not save any memory; by not using a JP instruction we eliminate, however, an absolute address which might normally need to be altered when re-locating (more of which in a later article). Another use for the three bytes of an entry is possible when 2 RSXs, using the same fundamental routine, occur together. The first entry space could be utilised to set a flag, to load a register with a value or to CALL another routine, any of which will apply to the first RSX only; the second entry must then JR or JP to or start the common routine. The routine above shows the 'proper' way of using the Command table - however part of the routine could be re-written as:

```
.comtbl DEFW namtbl     ;address of the name table.
.capon  LD   A,&FF      ;any non-zero value will set CAPS LOCK to on.
        DEFB &06        ;dummy instruction: when entered via .capon, this will
                        ;be seen and treated as LD B,&AF so using the next byte
                        ;(XOR A = &AF) also. In this way, register A holding
                        ;&FF will not be re-loaded with 0; the next instruction
                        ;to be obeyed is LD (&B4E8),A. There is also a saving
                        ;of 5 'T' states compared to the JR used previously.
.capoff XOR  A          ;zero will reset CAPS LOCK (ie to lower case): XOR A is
                        ;one byte (and 3 'T' states) less than the LD A,0 used
                        ;before.
        LD   (&B4E8),A  ;(&B632 for the 6128); another byte (and 4 'T' states)
                        ;saved by loading the flag directly instead of via
                        ;(HL).
        RET              etc
```

The entries in the Command table at .capon and .capoff now form part of the routines, so saving a further six bytes by eliminating two JPs. Note that three bytes are still allocated for the .capon entry in the command table - as there are no further RSXs after CAPOFF, the allocation of three bytes for this RSX is irrelevant.

All this alternative use of entry bytes may create one minor problem for anyone with a |HELP RSX which lists all RSXs and their routine addresses; these |HELPs work by picking the second and third bytes of each entry in the Command Table & printing this as an address. If we have used these bytes in a different manner, the address so printed will be meaningless.

The Command Name table consists of the list of the names we have chosen for our new RSX Commands (the | character which precedes the RSX command in BASIC does not form part of the actual name and should not be present in this table). Each name needs to start with a letter and may contain letters or digits or the full stop character only with a maximum of sixteen characters. All letters must be in upper case and the last character of each name has to have bit 7 set (ie. must have 128 or &80 added). Each name follows immediately after the end of the previous one & the last name must be followed by a byte of &00 to indicate that it is the end of the list.

Once the RSX has been initialised. it should not be re-initialised - doing so would alter the data in the Data Block and result in a crash!!! The easiest method to avoid this is to insert a byte of &C9 (machine code RET instruction) as the first byte of the initialisation routine after this has been used once.

Groups of RSXs can be initialised at any time; each such group will set up its own block of data which chains into those already there.

ERROR TRAPPING

A decision has to be made as to whether it is OK to just exit the RSX without doing anything should an error occur (eg the number of parameters is wrong, as in the |CAPS RSX at the end of the this article) or whether an error message needs to be printed & the BASIC program stopped to draw attention to the fault.

The first option allows a simple RETurn from the routine (assuming nothing has been PUSHed onto the stack) while the second can be facilitated using routines in ROM. If the second way is chosen, we can use BASIC's error handling routine which can report any of the usual BASIC error messages, or if none of these is suitable, we could print our own before using the same exit path.
This exit method is useful in that it does not require that anything PUSHed on the stack must first be POPped, and this also gives an ERR number which can be 'trapped' if needed. We can even arrange that our error has its own unique ERR number. In the routine below is a suggested error routine starting at .parerr.

Next time we shall look at RSXs which need data fed to them to work and in turn are able to output data also. To end with, here is another version of the CAPSON and CAPSOFF routine which incorporates most of the points given so far and including a couple which will be explained next time. It has been reduced to the single RSX, '|CAPS' which requires the flag value to be input directly with it using the syntax:

```
|CAPS ,<Flag value (0 for lower case; non-zero for upper case)>
```

```
        ORG  &8000
.intrsx NOP             ;this byte will be filled with the RET instruction
                        ;byte to stop re-use of this initialisation routine.
.datblk LD   HL,&8000   ;the address of the NOP byte. Also, the 4 byte data
                        ;block used by KL LOG EXT starts at the first byte
                        ;of this instruction.
        LD   (HL),&C9   ;store a RET byte.
        INC  HL         ;HL now points to .datblk for KL LOG EXT.
        LD   BC,comtbl  ;address of the command table.
        JP   &BCD1      ;return directly to BASIC after Firmware Call KL LOG EXT

.comtbl DEFW namtbl     ;address of the name table.
.caps   DEC  A          ;now zero if Flag value parameter only is present.
        JR   NZ,parerr  ;if no parameter or too many, then report error.
        LD   A,(IX+0)   ;get the Flag value.
        LD   (&B4E8),A  ;(&B632 for the 6128) Set the system variable with the
                        ;value input.
        RET
```

(continued on next page)

(continued from previous page)

```
.namtbl DEFM "CAP","S"+&80
        DEFB &00        ;end of Name Table byte.

.parerr LD   HL,errmes ;HL -> error message 'Check Parameters'.
        CALL prtstr    ;print the error message.
        LD   A,33      :ERR number; anything above 32 is an 'Unknown Error'.
        LD   C,0       ;for ROM 0 (BASIC).
        LD   HL,&CA93  ;(&CB55 for 6128); BASIC's error handling routine in ROM
        JP   &001B     ;to FAR CALL to ROM; the error handling routine alters
                       ;the stack so doesn't REturn.

.prtstr A,(HL)         ;get first/next char. of string to print.
        OR   A
        RET Z          ;if string finished.
        CALL &BB5A     ;print char.
        INC  HL        ;step on to next char.
        JR   prtstr    ;for next character.

.errmes DEFB "Check Parameters".&00
```

TECHNICAL TERMS

In the meantime here is a brief explanation of 'T' states. The rate at which the microprocessor carries out any Machine Code instruction is measured in 'T' states. The number varies from instruction to instruction: eg XOR A requires 4 and LD A,0 needs 7, showing that XOR A is faster than LD A,0 by 3 'T' states.

A 'T' state is actually a clock pulse. So that if the computer's clock driving the microprocessor runs faster (upto a maximum specified for the microprocessor chip), the instructions are processed faster. However, while a 'T' state is a clock pulse, the reverse is not necessarily the case. This is because of things called 'Wait' states. These are clock pulses that the microprocessor is told to ignore - eg. when accessing slower memory or when Direct Memory Access (DMA) is required by an external system. Any such Wait states are additional to the specified number of 'T' states an instruction may require: for example there might be 5 clock pulses occurring during the processing of the XOR A - 4 'T' states plus 1 Wait state. The actual number of pulses that are required for the XOR A will still be at least 3 less than those required for LD A,0.
In the CPCs, although the system clock operates at 4Mhz (that's 4 million clock pulses per second), there are restraints on memory which reduce the number of usable pulses to about 3 million per second by interspersing 'Wait' state clock pulses.

SCULL PD IS BORN

In our last issue, there was an article about CPD - one of the few tape based Public Domain libraries for the CPC. It has now expanded to include disc based PD software and to explain the changes we asked Alan Scully, the master-mind behind it, to write a follow-up article.

After my article in the last issue of Print-Out, I have expanded my PD library enormously, and have achieved success in getting mentions in the major Amstrad magazines. The PD library also now supplies PD on Disk as well as Cassette and because of this the library is now called Scull PD (it would be a bit silly to call it Cassette Public Domain now that I do disks!!). There are a lot of good programs in my PD library, you may go Wild over DW EasyDos DeskTop, go crackers over the Playable Demo of TOTAL ECLIPSE (thanks to Incentive Software), or you may even be impressed by Woboll - Thomas Defoe, editor of this superb magazine, was. Of course, with over 300 programs on disk & 250 on cassette, there's sure to be something of interest in the library!

You might remember in issue 4 of the mag, Tony Kingsmill was selling adventure games, and they got quite good reviews. Tony recently sent me a copy of ESCAPE FROM THE ALIEN SPACESHIP, another one of his adventures (about a third the size of his adventures for sale) & has given me permission to use it in my library, the program can be found on PD CASSETTE 15, which also contains 5 other games, or on PD DISK 8 which contains lots more programs including the Total Eclipse playable demo.

Anyone who buys Amstrad Action will notice that David Wild's PD programs have been given a lot of coverage. Unfortunately, these programs were available on disk only, but now Scull PD can bring them to you on cassette (well, most of them). PD CASSETTE 10 contains nine of his best programs (inc BASIC+, Hackers Basic, and Pilot CPC), and some of his other programs are 'dotted' about the library, for example, MiniCAD is available on PD CASSETTE 16.

From some of the letters recieved, assemblers seem to be in demand. There are two assemblers in the library; AA Micro Amstrad Assembler+ & Power Assembler. Both these programs support many commands and are very easy to use. PD DISK 7 contains both assemblers, PD CASSETTE 16 contains Micro Amstrad Assembler+, & PD CASSETTE 11 contains Power Assembler.

Almost everybody has bought a computer game, and almost everybody has games that they feel that they need a little 'help' with, usually in the form of a poke. PD CASSETTE 13, contains loads of pokes to help you on your way, all of which are written by Hack Master Andrew Price. If there's a game you need help with then order Hack Attack now - it may just have the poke your looking for.

If anybody out there has written any programs, or has programs from other PD libraries (WACCI CP/M only, and Robot PD selections AMS10 onwards and CPC13 onwards), then PLEASE send them to me and I will send you some PD in exchange (ie I will not charge for it). I plan to get as much PD as possible, so please send any PD you have (no magazine listings).

That's about it for this issue, all that's left to do is fill you in on a few of the library details and to show you the latest pack list.....so here goes!

* For any TWO selections from the PD CASSETTE range send a C15 cassette, 50p, and a Stamped addressed envelope.
* For any TWO selections from the PD DISK range send a disk, £1, and a Stamped addressed envelope.
* To get your PD free, send some PD with your order & I will return your money if I use it.
* All orders are dispatched the same day (where possible).
* Stock List regularly updated, send SAE for complete list.
* Scull PD Library - The quickest, the cheapest, the best, the largest non-CP/M range (CP/M PD soon), You'd be off your Scull to go anywhere else!
* Scull PD Library can be contacted at (if phoning, ask for Alan) :-
  119 Laurel Drive, Greenhills, East Kilbride, Glasgow G75 9JG. (03552) 24795

PD CASSETTE 1 - GAMES 1	PD CASSETTE 8 - STARTER PACK	PD CASSETTE 15 - (BIG) GAMES 6
PD CASSETTE 2 - GAMES 2	PD CASSETTE 9 - GAMES 4	PD CASSETTE 16 - APPLICATIONS 3
PD CASSETTE 3 - (BIG) GAMES 3	PD CASSETTE 10 - PROGRAMMING	PD CASSETTE 17 - SERIOUS 2
PD CASSETTE 4 - SUBROUTINES/ROUTINES	PD CASSETTE 11 - APPLICATIONS 2	PD CASSETTE 18 - SERIOUS 3
PD CASSETTE 5 - AI/EDUCATIONAL	PD CASSETTE 12 - GAMES	PD CASSETTE 19 - (BIG) GAMES 7
PD CASSETTE 6 - APPLICATIONS 1	PD CASSETTE 13 - HACK ATTACK 1	PD CASSETTE 20 - SERIOUS 4
PD CASSETTE 7 - SERIOUS 1	PD CASSETTE 14 - GRAFIX	PD CASSETTE 21 - TOTAL ECLIPSE DEMO
PD DISK 1 - SERIOUS 1	PD DISK 5 - DW DISK 1	PD DISK 9 - SERIOUS 2
PD DISK 2 - GAMES 1	PD DISK 6 - DW DISK 2	PD DISK 10 - GAMES 4
PD DISK 3 - GAMES 2	PD DISK 7 - APPLICATIONS	PD DISK 11 - ARTIFICIAL INTELLIGENCE
PD DISK 4 - ANIMATION DEMOS (6128)	PD DISK 8 - GAMES 3	/EDUCATIONAL/GRAPHICS

What PRINT-OUT thought

Despite the lack of paper documentation with these cassettes/discs the programs themselves give you enough information, in most cases, to successfully operate the routines and in some cases you are even told how the program works. Prices are reasonable enough - especially for the tapes - and the service is usually quick and reliable. Some of the programs are of really high quality and would not shame a homebrew collection and most programs, although often quite short, perform a useful task (some more obscure and less valuable than others). There seems to be, in general, a large selection of programs in the library and they are all quite varied. One of the drawbacks with any PD library is that you are never quite sure of what you are going to get. However, with Scull PD you are likely to find something that is suitable and useful but some of the routines come from magazines or other PD libraries and so, in a few cases, you may find yourself duplicating what you have already received from other sources. Each tape holds, on average, about 13 programs and each disc has about 20 programs (although these figures do vary considerably). Use of the discs is much easier than tapes as all the programs can be easily located. Each disc also comes with 'DiskTop', a very useful program which allows you to RUN, LOAD, RENAME & ERASE programs as well as a few other options such as catting discs.

Advanced BASIC

Customising Input Routines

It is often said that in order for a commercial program to be a success, it needs to be as user-friendly and helpful as possible. As the time when the user needs to interact most with the computer is when information is being inputted, it would seem sensible to make this part of the program as simple and easy-to-use as possible. In this issue's Advanced BASIC section, we'll be concentrating on using the CPC's inputting commands to the best of their ability.

Almost every BASIC programmer knows about the INPUT command, but for those of you who don't, type this in :-

```
INPUT number
```

Helpful, isn't it? All it does is print a question mark and then wait for something to be entered. As the variable following it, number, is numeric it wants a number to be inputted. However, if we did not know that it required a number and entered a letter instead, the computer would respond with ?Redo from start - not very useful. In order to tell the user what the computer needs, we could print some text as well. Try this :-

```
INPUT "Enter a number : ",number
```

This is more friendly but still throws out an error message if a letter is entered by mistake. One way round this, is to change the numeric variable to a string variable by entering :-

```
INPUT "Enter a number : ",number$
```

This will now except both letters and numbers, not very useful if ONLY numbers are meant to be allowed. To solve this problem all we've got to do is to break down the string into one character long strings and then see if any of them are letters. If so, the entry is disregarded and the program repeated. The following program will do this :-

```
10 INPUT "Please enter a number : ",number$
20 word=0:b=LEN(number$)
30 FOR i=1 TO b
40 c$=MID$(number$,i,1)
50 valid$="1234567890"
60 check=INSTR(valid$,c$)
70 IF check=0 THEN word=1
80 NEXT i
90 IF word=1 THEN GOTO 10
```

Lines 30-80 split the entry up into strings just 1 character long (Line 40) & then check to see if the character is allowed (Line 60) and if it is not, the flag 'word' is made equal to 1 to indicate that a letter is present (Line 70). The characters which are allowed are stored in valid$ (Line 50).
Although this works it can present a slightly untidy screen display (especially if a very long word was entered) and to overcome this the following alterations need to be made :-

```
 1 MODE 2
10 LOCATE 1,1:PRINT "Please enter a number : "
11 LOCATE 25,1:INPUT "",number$
90 IF word=1 THEN LOCATE 25,1:FOR i=1 TO b:PRINT " ";:NEXT i:GOTO 11
```

These modifications are fairly self explanatory. Line 1 is needed in order to tidy things up & line 10 simply prints the text at the required position on the screen - it doesn't input anything because the text printing and inputting commands need to be on separate lines. Line 11 does the inputting - the LOCATE command positions the cursor at the end of the text and the "", is required so that a question mark is not printed. Line 90 then erases whatever was inputted incorrectly and then goes back to line 11.
However, there is still one major problem: this program does not input a number but inputs a string which contains a number. The trouble with this is that you may not use it as a numeric variable or in any calculations. To illustrate this add the following two lines :-

```
100 PRINT "The number is : ";number$
110 PRINT "The number when multiplied by three is : ";number$*3
```

When the computer attempts to execute line 110 it gives up and prints 'Type mismatch in 110'. All is not lost though. The addition of these three lines....

```
 95 num=VAL(number$)
100 PRINT "The number is :";num
110 PRINT "The number when multiplied by three is :";num*3
```

....sorts the problem out. Line 95 converts a string to a numerical value which can be used in an identical way to any other numerical variable.

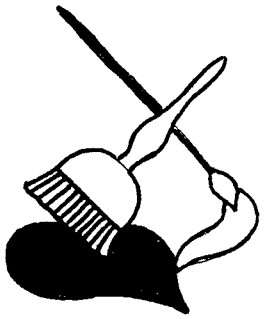
That's one method of inputting things, what about all the others? There are several other ways of getting information from the user by use of INPUT (eg the above routine could be modified so only words would be accepted) & also its companion, LINE INPUT. LINE INPUT behaves in an identical way as INPUT but accepts exactly what is typed (including commas) The other inputting command frequently used is INKEY$ and this was mentioned in Issue Three. With some programs it is necessary to allow a certain number of characters to be entered and the program below lets you enter a word and automatically stops when ten letters have been entered and it ignores the ENTER key :-

```
10 MODE 2
20 PRINT "Please enter a name of exactly ten letters : ";
30 a$=INKEY$:IF a$="" THEN 30
35 IF a$=CHR$(13) THEN 30
40 IF a$<>CHR$(127) THEN GOTO 100
50 b=LEN(name$):IF b<1 THEN 120
60 c$=LEFT$(name$,b-1)
70 name$=c$:PRINT CHR$(8);
80 PRINT " ";CHR$(8);
90 GOTO 120
100 name$=name$+a$
110 PRINT a$;
120 IF LEN(name$)=10 THEN PRINT:PRINT UPPER$(name$):END
130 GOTO 30
```

I will leave you to try and work out how it operates. The only clue is that the code for ENTER is 13 and that for DELETE is 127. It's important that, under certain conditions, various parts of the program are not executed and hence the large number of GOTO commands which all bypass sections of the lisiting. At the end, the name is stored in a variable called 'name$'. The program needs to recognise the DELETE key and act differently when it is pressed. The deleting part of the program is in lines 40 to 90 and the instruction PRINT CHR$(8) moves the cursor back one space on the screen. Even though it does not accept [ENTER], it still needs to recognise it so that the printing routine is bypassed, otherwise strange symbols might appear.

The program below is a modified version of that above. It will allow a maximum of ten letters to be entered but the entry can be stopped at any time, by pressing the ENTER key. Even when ten letters have been entered, it still lets you delete letters (so changing your entry) and this is terminated only by the use of the ENTER key. I hope that these comments have been useful and will have enabled you to make your own programs more professional and easy to use.

```
10 MODE 2
20 PRINT "Please enter a name of no more than ten letters : ";
30 a$=INKEY$:IF a$="" THEN 30
40 IF a$=CHR$(13) THEN PRINT:PRINT UPPER$(name$):END
50 IF a$<>CHR$(127) THEN GOTO 105
60 b=LEN(name$):IF b<1 THEN 130
70 c$=LEFT$(name$,b-1)
80 name$=c$:PRINT CHR$(8);
90 PRINT " ";CHR$(8);
100 GOTO 130
105 IF LEN(name$)=10 THEN GOTO 30
110 name$=name$+a$
120 PRINT a$;
130 GOTO 30
```



COLOURDUMP 2

Colour printing with an Amstrad

If you mentioned that your computer was able to print colour graphics using a standard printer at very little extra cost, the chances are that people would think that you were either mad or had invented a new programming technique. But neither is strictly true. It's possible to print full colour screen shots & any other graphics that you choose for only £12.95 plus an Epson compatible printer.

Before I get into the review proper, I will make a few general points about Colourdump 2 and the printers that can be used with it. Colourdump 2 is written exclusively for MJC and can be bought only through them and costs £12.95 and is available only on disc. Although the advert for it says that it works only on a CPC 6128, I have found that it operates perfectly on a CPC 464 + Disc Drive. It will work on any printer that is Epson compatible, and this includes almost all 9 pin dot matrix printers including Amstrad DMP2160, DMP3000, DMP3160, DMP3250, Star LC-10 (mono and colour), Citizen 120-D and Panasonic KXP1080/KXP1081. With a Star LC-10 colour it produces its best results as all four colour ribbons can be contained in one ribbon cartridge. With normal black and white printers, you need to buy four coloured ribbons (or colour carbon paper) - black, blue, red & yellow. 24 pin printers can also be used but with these, the print quality will be worse than with a normal 9 pin printer. Enough of the technical comments and on with the review.

As well as the disc, you receive a professionally presented manual which is set out in a very clear and easy-to-understand manner. This contains any introduction to the program which sets the background to the program & this is followed by a section which describes the various machines that can be used & also tells you what type of screens you can print. The program can print out OCP Art Studio and AMX Art pictures along with the normal screens that you create using

Although we are unable to reproduce the colour of the print-out, below is a screen shot printed by Colourdump 2. This is the actual size of the picture and shows the quality of printing using the program and a Star LC-10 Colour.

BASIC or Machine Code. The main limitation of the program becomes apparent here, it is unable to print out screens saved by Multiface devices. However, this is not strictly true as it is possible to convert a Multiface screen into a normal screen using a utility in a recent issue of Amstrad Action. Providing that the screen has not changed size, you can print this out in the normal manner. The second problem with the program is that you have to know what the colours are - a major difficulty if you have a green screen, although it can be overcome with a bit of patience.

The program is very clearly set out and takes you through printing a screen step-by-step and a test screen is included on the disc for you to print out. It is the manual which is the program's strong point. For without it, this program could have become impossible to use but, as it is, it's extremely easy to start printing pictures. The program includes many advanced functions such as setting the shadings but these are invisible until you want them to.

Overall, it is a very well thought out program and makes a worthwhile addition to anyone's utilities library. Highly recommended for anyone who has a dot matrix printer - an absolute bargain at the price.

Homebrew Idea

I have an idea for a new column in Print-Out. As you review homebrew software as well as programming I feel that a Help and Tips page for homebrew software would be a good idea. The column could include tips for serious & educational software, as well as help with games (maps,pokes,passwords.etc.). If you consider this, I'm sure that many readers would make contributions, as well as the writers of the software. I feel this is valuable for many homebrew companies, being as larger CPC magazines don't really cover this field of software.

TONY KINGMSILL. ST ALBANS

P-O: We're always willing to consider any new ideas so if you have any suggestions for the magazine, write in and let us know. As for the idea above, we need our readers to tell us what they think before we start a new section. It's very true, however, that homebrew software is given very little coverage in national magazines and we are keen to encourage all authors as much as possible.

How about.....?

And now two quick questions - 'Have you thought of producing a cover cassette?' from ALAN SCULLY, GLASGOW and 'How are people supposed to know about your next issues of Print-OUT?' from CHRIS RUSSEL, HARROW. The answers to these questions are, 'Yes we have, but it would mean doubling the cost of the magazine & so it probably will not happen' & 'We advertise each issue of Print-Out in the Small Ads section of Amstrad Action approxiamately once every two months'.

This article was written by Richard Sergeant and is hopefully the first part in a larger series which will continue into future issues. Many people neglect the CPM disc &, as the CPC is quite remarkable in the fact that it can operate CPM, we printed this article in an attempt to let users get the most from their CPC.

Getting to grips with CPM+

The reason I am writing this little article, which may continue into some form of a mini series is to encourage CPC users to use CPM Plus or, for the less fortunate, CPM 2.2. I intend to take you through some simple exercises and examples to demonstrate that CPM does work and I personally consider it to be a far superior operating system than Amsdos for the serious (business) user.

FOR EASE OF UNDERSTANDING WHAT IS REQUIRED WHEN ENTERING IN INSTRUCTIONS, CAPITAL LETTERS HAVE BEEN USED FOR THE COMPLETE WORD OR STRING OF WORDS.

Often, when the first time user puts the CPM Plus System disc (but when I say System disc I mean a Copy and not the Master) into the machine and types |CPM, he or she does not know what to expect. After a little disc activity the sign on Copyright message appears and then the all important A> prompt. Should the user now try to move the cursor using the cursor keys all that is produced is continental characters. What we have to do is configure the keyboard, and the utility to use is SETKEYS.COM.

If we now enter SETKEYS (as the COM extension is not needed) and press <ENTER> the computer should respond with 'Cannot open file', not very helpful you must agree. We should have typed SETKEYS KEYS.CCP now when we press <ENTER> nothing appears to happen, we are just back at the A> prompt. However, we now have use of the the left and right cursor keys and the <CLR> key works correctly.

If you want to look at the contents of KEYS.CCP, enter TYPE KEYS.CCP and a list of special control characters will appear on the screen. If you now load this file into your friendly word processor - PROTEXT in PROG mode is ideal - it is perfectly possible to alter it to configure the keyboard to many useful combinations to suit yourself - however, writing new SETKEY files could well be the subject of a separate future article. To get a print out of everything that is echoed to the screen, hold down the <CTRL> key and press 'P'. A beep will be emitted when activated and pressing <CTRL>+'P' again stops printing.

If for some reason you are a lousy typist and cannot spell and had entered SATKEYS KEYS.CCP, the computer would respond with SATKEYS? (another less than helpful comment). The question mark is CPM's way of telling you that it can't

find any file in the disc's directory which is called SATKEYS. A quick way to edit your mistake is to hold down the <CTRL> key and press 'W'. The last line you typed is repeated, now using the cursors keys edit in the normal way, and when you are happy with your entry press <ENTER> again.

All the examples in this article will work on CPM Plus and some may also work under the less advanced CPM 2.2 (which is supplied with the DDI-1 disc drive).

> For the information of all 464 and 664 owners, CPM + will work on your machines, but you do need the dk'tronics 64K expansion pack, a copy of the specially patched version of C10CPM3.EMS (the disc based file that makes up part of the operating system) and of course a DDI1 disc drive (664 owners, are there any left, don't need the disc drive).

We can see that many CPM programs need a parameter after the file name. Try this, PALETTE 63 0 - the screen colours are now reversed. To return the screen to the default settings, type PALETTE 0 63. For all those (un)lucky people who have colour monitors try PALETTE 21 0 - this, I consider, will make the screen easier to read. Owners of Mono monitors will in general, find using CPM kinder on the eyes because most professional programs are done in Mode 2. Mind you on the CPC it is perfectly possible to have programs running in Modes 0 & 1 under CPM, DISCKIT3 is an example of a Mode 1 file (to be truthful DISCKIT3 is not a true CPM file as it jumps back into Amsdos so that all available memory can be used when copying discs).

It is time to try out a few more commands. Type RENAME and the program will respond with 'Enter New Name:'. At this prompt type PROFILE.SUB and then press <ENTER>, the program now asks 'Enter Old Name:' now type PROFILE.ENG and press <ENTER>. If we type DIR[FULL] we're presented with a complete catalogue of the files on the disc and can see that PROFILE.ENG has disappeared and PROFILE.SUB is in its place.
Renaming files can be shortened to REN PROFILE.SUB=PROFILE.ENG, and once again if we enter TYPE PROFILE.SUB, a short list is presented which should look like this:

```
A>type profile.sub
setkeys keys.ccp
language 3
```

This is what is called a batch file and as long as a file called SUBMIT.COM is on the disc every time you initialise CPM with |CPM the two files SETKEYS & LANGUAGE will be run automatically. The '3' behind LANGUAGE indicates that the the keyboard and screen are set up for the UK pound sign whenever the hash '#' or '£' keys are used. Once again, this PROFILE.SUB file is easy to alter using a text editor or word processor and contains all sorts of instructions. On my SuperCalc2, PCW 9512 Start of Day disc, I have the PROFILE.SUB containing the following lines:

```
  date set
device lst:=par
setdef m:,a:[order=(sub,com) temporary=m: display]
matrix a4
palette 1 0
language 3
setkeys keys.wp
setlst ukset.lst
pip
<m:=dattim.com[r]
<m:=sc2.com[r]
<m:=sc2.hlp[r]
<m:=sc2.ovl[r]
<m:=*.xqt
<
dir a:work12*.cal[full]
m:
dattim
sc2 start
<a
a:
cpmkeys
basic rped
```

This just goes to show you the complexity that can be achieved. The only thing that I have to enter is the date and time when asked. Two of the files in the above list that do not work on CPCs are MATRIX.COM & CPMKEYS.COM. The other two, BASIC.COM and RPED.BAS, were supplied to PCW users, but with a little bit of adaptation they will work happily on the CPC machines.

And below are two more examples of SuperCalc2. The one of the left is as used on a CPC which is fitted with a double density 3.5 inch 'B' drive. The one in the box on the right, is a PROFILE.SUB file to auto run the same program on a single drive CPC.

```
ramdos+ d1
setdef b:,a:[order=(sub,com) temporary=b: display]
b:
date set
language 3
setlst ukset.lst
setkeys keys.sc2
palette 63 0
paper f66,16,g4,c,p on
dattim
dir work12*.cal
sc2 start
<A
language 0
setlst usa_set.lst
setkeys keys.ccp
basic rped
```

```
date set
language 3
setlst ukset.lst
setkeys keys.sc2
palette 63 0
dattim
dir work12*.cal
sc2 start
language 0
setlst usa_set.lst
setkeys.ccp
```

You will have noticed that I have used a number of utilities but I have not explained about their uses. Well I've decided to hold it over until next time, just to gauge the feed back about the usefulness of this article.

In conclusion, I do hope that I've encouraged some of you to dust off those System discs and have a dabble with CPM. Should you progress into the world of CPM Plus, it can offer software such as SuperCalc2 - probably the best spreadsheet for the Amstrad CPC's and PCW's. Databases like DBase2 and Condor1, and word processors such as CPM+ Protext and Newword 2 - if you use large capacity drives, these WPs will allow the creation of data files of over 200K in length, which should be enough to be going on with for most users!!!

So there you have it folks. Let me know what you think. Do readers of PRINT-OUT want to read this sort of article, I look forward to your comments.

SMALL ADS

FOR SALE - AMSTRAD CPC 464 INSTRUCTION MANUAL as new, £8 + £1 postage, or will exchange for four 3" discs. P.H. Breckin, 161 Longsight Road, Holcombe Brook, Bury BL8 4DA. Phone 0204-88-3443.

FOR SALE - Over 75 Amstrad games. Mostly budget and compilations. Cost over £150 to buy, sell for £35. Phone Alan (03552) 24795.

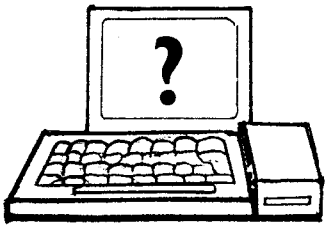
FOR SALE - Unwanted Amstrad games in perfect condition - Wizball £1.00, Super League £1.00, Peter Beardsley's International Football £1.00, or buy the lot for £2.50. T. Kingsmill, 202 Park Street Lane, Park Street, St Albans, Hertfordshire AL2 2AQ.

HOMEBREW - Homebrew adventures for sale - send for a list. T. Kingsmill, 202 Park Street Lane, Park Street. St Albans, Hertfordshire AL2 2AQ.

2 MAGIC GAMES for only £3. Rebound and One Arm Bandit Simulator. Send cheques/Postal Orders to Alan Scully, 119 Laurel Drive, Greenhills, East Kilbride, Glasgow.

PUBLIC DOMAIN software for Amstrad CPC on cassette. For anyone who missed the article on CPD, you can receive PD for only 25p and a blank tape (& a SAE). Alan Scully, 119 Laurel Drive, Greenhills, East Kilbride, Glasgow G75 9JG.

see p.45 for more details



TECHNICAL TIPS

Over the past two months we have received quite a few letters seeking advice and assistance with various problems concerning the CPC. Some of them, we feel, will apply to many users and so we have decided to print the solutions to them in 'Techincal Tips'.

Q. The first problem comes from TONY KINGSMILL of ST ALBANS who says :-

'I would be very grateful if you could tell me a way to turn off a ROM from BASIC or Machine Code. The reason for this is that I bought a speech ROM a few months ago and have found that some games are not compatable with it. Therefore if you know a way so that I don't have to keep removing it I would be grateful.'

A. It is a well documented fact that some ROMs interfere with certain programs and one of these is the Dk'Tronics Speech Synthesiser ROM. The solution to this problem is to switch the ROM off; however, in order to avoid having to continually remove and insert the ROM chip (not a good idea) you need a program to do this. Some ROMs include a | command to switch themselves (& any other ROMs off) but certain ones do not and for this reason we have written 'ROMSWITCH' which allows you to turn off any particular ROM(s) or all the ROMs that are attached.

```
[F1] 10 'ROMSWITCH-LOADER copyright R Taylor 1990
[B3] 20 RESTORE:PRINT:PRINT"Please wait a few seconds"
[16] 30 FOR lin=0 TO &48/8-1:total=0:FOR n=0 TO 7:READ a$
[F8] 40 byte=VAL("&"+a$):POKE &BF00+lin*8+n,byte
[4B] 50 total=total+byte:NEXT n
[21] 60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT"Error in line"lin*10+110
        :END
[C4] 70 NEXT lin
[BB] 80 PRINT:PRINT"All M/C loaded":PRINT:PRINT"Press 'S' to save M/C as ROMSWT
        CH.BIN":WHILE INKEY$="":WEND:IF INKEY(60)<>-1 THEN SAVE "ROMSWTCH.BIN",
        B,&BF00,&48
[4D] 90 PRINT:PRINT"To Load and use ROMSWITCH just Enter:":PRINT"LOAD"CHR$(34)"
        ROMSWTCH.BIN"CHR$(34):PRINT"THEN Enter:":PRINT"CALL &BF00 [,list of ROM
        s required to be on]"
[EA] 100 END
[30] 110 DATA 21,43,BF,77,B7,28,0C,47,2CC
[D5] 120 DATA 23,DD,7E,00,77,DD,23,DD,3D2
[41] 130 DATA 23,10,F5,0E,00,21,1B,BF,231
[71] 140 DATA CD,16,BD,11,40,00,21,FF,311
[40] 150 DATA AB,3A,43,BF,B7,28,11,47,31E
[E7] 160 DATA DD,21,44,BF,C5,DD,4E,00,3F1
[48] 170 DATA CD,CE,BC,DD,23,C1,10,F4,51C
[E2] 180 DATA 3E,C9,32,CB,BC,DF,40,BF,49E
[49] 190 DATA 06,C0,00,03,07,06,02,00,0D8
```

Before running the program you should save it and then follow the instructions. When RUN the program will save the machine code as ROMSWTCH.BIN so that for use in the future just type LOAD "ROMSWTCH.BIN" and follow the instructions below. The routine is now ready to use by CALLing &BF00 followed by the numbers of the ROMs that you want left ON - all other ROMs will be disabled. eg CALL &BF00,2,5 will turn off all the ROMs except those that are in slots two and five.
The list of ROMs required should be in ascending order so that their reinitialisation will be in the order that the computer usually uses. Every ROM number should be separated from its neighbours and from the CALL &BF00 by a comma. If no ROMs are required, then just use CALL &BF00.
NB that this Machine Code routine will RESET the computer and any program that is present will be lost.

Q. The next query comes from I.L.BYERS of LONDON who writes :-
'I am the owner of a 6128 and on 2 discs am unable to either RUN the programs or copy or format them. No mention in the manual to overcome the error message:'

DISC ERROR
TRACK 0 SECTOR #4

A. There is indeed no such message present in my AMSDOS ROM, although there is an undocumented 'Disc Fail' one hidden away at &CC78. Neither is there a string of 'Track' or 'Sector in the ROM so I can only assume that the DISC ERROR message is coming from some program that you are running. As to the problem itself, it seems to me that you have a couple of damaged discs, which might be noticeable if you look at the surface while rotating the disc with the shutter pulled back (taking care not to touch the actual surface) It should be possible to transfer most of the other tracks and sectors from the 2 discs to new ones using a good Sector Editor. If you would like to send us the two discs and two new ones, we would be willing to try and do that for you.

Q. Moving on we come to a quick query concerning light pens from ALAN SCULLY :-
'I would like to enquire as to whether you would happen to know the wiring code for the 'Electric Studio' light pen. I had a slight accident with mine; all the wires came out of the top (it is a short story but I will not bore you with the details), anyway, I would dearly like to get the light pen working again.'

A. Unfortunately, I do not own a light pen but if any of our readers know about it or have a copy of its wiring code, please send it in. My advice would be to get in touch with the manufacturers as they should be able to give you the information that you need.

COMPARING WORDS & Acting on the outcome

Machine Code

Last issue, we looked at making decisions in Machine Code by using the compare (CP) command and the flags. You will have noticed that the compare instruction can only work with numbers and, indeed, the flags are set according to results of numerical calculations. You might be wondering, therefore, how you can make decisions which are based on words. Fortunately this can be done but, however, it is not quite as simple as using an IF...THEN...ELSE command in BASIC.

You'll have read, in the first part of this tutorial, that we have to input a word & then see if it is the same as one that's held in memory. We've already worked out an inputting routine which gets a word from the user, and puts it in memory at an address labelled .store. Before we go any further, it is important to remember that the word will be held in memory as a series of numbers and not letters. These numbers will be the ASCII codes of the letters that they're representing. Thus the word AMSTRAD will be held in memory as the decimal numbers, 65,77,83,84,82,65,68 or the hexadecimal numbers 41,4D,53,54,52,41,44.

Therefore, a way to see if two words are the same is to compare the numbers that make up the words, & this is exactly what our program will do. The routine will need to know two things before it is called and they're the address of the word that has been inputted, and the address of the word with which it is to be checked. The former address should be loaded into DE and the latter into HL.

Then, when the routine is called, A will be loaded with the contents of the address pointed to by HL & this will be stored in B temporarily. A will then be loaded with the contents of the address pointed to by DE. At the end of this, B will hold the ASCII code of the first letter of the checking word & A will contain the ASCII code of the first letter of the inputted word. Next, a normal CP instruction checks to see if the value in A equals the value in B - if it does then the zero flag will be set (since A minus B will equal zero if A and B are the same). The instruction JP NZ,wrong is only executed if the zero flag is not set (ie. A and B are not the same - A minus B does not equal zero). The subroutine, '.wrong', will then perform whatever action is necessary for an incorrect answer. But, if the program does not jump to .wrong, the routine adds 1 to both HL and DE so that they're pointing at the next letter in the two words. A check is now carried out in order to see if A equals zero (the end of word mark) & if it does, this means that the end of the word has been reached without an error occuring and so it jumps to the label, .right. If A doesn't equal zero, the end of the word has not been reached and so the comparison process is repeated.

The program below will not work on its own because we have not inputted a word or included any subroutines for what happens when the words match or are diff-

erent. The reason that it is printed, is because it can easily be incorporated into any program of your own, providing you have assigned correct values to HL and DE as explained before.

```
        ORG &4000       ; program located at &4000
        LD HL,test      ; HL holds the address of the inputted word
        LD DE,store     ; DE holds the address of the word to check with
        CALL check      ; call the checking routine
        RET             ; return to BASIC

.check  LD A,(HL)       ; load A with the value contained in address HL
        LD B,A          ; B equals A
        LD A,(DE)       ; load A with the value contained in address DE
        CP A,B          ; see if A equals B (FLAGS = result of A-B)
        JP NZ,wrong     ; if A is different from B (zero flag not set)
                        ; jump to the subroutine called .wrong
        INC HL          ; HL=HL+1
        INC DE          ; DE=DE+1
        CP A,0          ; see if A equals 0 (FLAGS = result of A-0)
        JP Z,right      ; if A equals 0 (zero flag set) the jump to the
                        ; subroutine called .right
        JP check        ; otherwise jump back to the beginning
```

All that remains for us to do is to put together all of the various modules that we have written to produce a proper program which works. The example below does this by asking a geographical question and getting the answer and checking it. This is the first completed program that we have looked at which does something useful. From now on we will be writing far more complicated programs and won't be just sticking to text either. Hopefully, you are now writing programs of your own but don't worry if there not very good as Machine Code is extremely difficult to learn to program competently and takes time, patience and a little bit of luck.

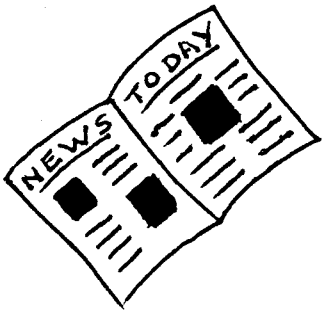
As a challenge for you, when you have run the program below and worked out how it works try and add more questions and turn it into a simple Machine Code testing program for anything that you fancied; all you need to do is change the questions and answers. Once you have done that, try adding a scoring system.

Regular readers will have noticed that we have not included any numbers for use with the BASIC Poker. The reason for this is that due to the large quantity of numbers that would have to be used to enter the text, it would be too long & could not be printed next to the lines to which they refer. However, next issue we will hopefully include them again.

I hope that you have found this article interesting and informative but if there's something else that you'd like to know, please feel able to write to us with your query. Until next time, have fun !!!

THE COMPLETE LISTING

```
ORG &4000
LD HL,mess1
CALL print
LD HL,store
LD B,0
CALL input
LD (HL),0
LD HL,test
LD DE,store
CALL check
RET

.input
CALL &BB06
CP 13
RET Z
CP 127
JP Z,erase
CALL &BB5A
LD (HL),A
INC HL
INC B
LD A,B
CP 15
RET Z
.dummy
JP input
.erase
LD A,B
CP 0
JP Z,dummy
DEC B
DEC HL
LD (HL),0
LD A,8
CALL &BB5A
LD A,32
CALL &BB5A
LD A,8
CALL &BB5A
JP dummy

.check
LD A,(HL)
LD B,A
LD A,(DE)
CP A,B
JP NZ,wrong
INC HL
INC DE
CP A,0
JP Z,right
JP check

.wrong
LD HL,mess2
CALL print
RET

.right
LD HL,mess3
CALL print
RET

.print
LD A,(HL)
CP 0
RET Z
CALL &BB5A
INC HL
JP print

.mess1
DB "This program tests your Geography. What is the
capital of Spain ?",0
.mess2
DB 13,10,"WRONG. The capital of Spain is Madrid",0
.mess3
DB 13,10,"CORRECT. The capital of Spain is Madrid",0
.test
DB "Madrid",0
.store
```



News & Views

It has been quite an interesting couple of months as far as matters Amstrad are concerned both in the large scale and also the small companies which Print-Out tries to encourage.

New AMSTRAD CONSOLE

Now that the CPC 464 is six years old, the software houses' seem to be very interested by the prospect of a new range of CPC computers, including the often promised console version. Despite the deliberate lack of information concerning the new machines, many people are becoming very intrigued by what the computers will be capable of and what the differences between them and their predecessors are going to be - none more so than existing CPC owners, who are worrying as to whether they will be left out on a limb by Amstrad and software developers with the imminent arrival of these 'super-CPCs'.

However, as has already been seen with the CPC 6128, software companies are often reluctant to take advantage of the higher capabilities of upgraded models and so will want to write software that will work on the widest number of machines. Admittedly, programs will be written specifically for these 'super-CPCs' and, in the sudden excitement of the new releases, software houses will be very anxious to gain a lead over their competitors as early as possible.

Amstrad themselves are treading a very fine line between keeping supremacy in the 8-bit market and badly hurting the existing CPC range. The main question is whether Amstrad intends the new CPCs to be replacements or additions to the 464/664/6128. If it is the former, you cannot help but think that maybe they've left it too late - after all, nobody could deny that the Amiga and other 16-bit computers are superior in terms of capabilities to the CPC. However in my opinion the CPCs still represent the best value-for-money & they have a lot of life left in them. So therefore Amstrad will want to keep the large CPC user base (& its software) and at the same time provide a new alternative to the 16-bit computers of today.

When you consider that over 2 million CPCs have been sold, it seems fairly stupid for Amstrad to discontinue it; even if Amstrad did, the CPC wouldn't die for it has so many users that software would continue to be produced & a whole host of small companies would spring up, all run by users. The BBC Micro is an excellent example of this, despite the lack of support by its manufacturers for several years now, it still has a magazine every month and a limited amount of software is produced for it.

Until Amstrad announce the actual details of the machines later this year, probably around September, the rest of us will have to wait to hear what these computers are able to do. One thing is certain, Amstrad's announcement on its console and 'super-CPCs' will ultimately affect the future of our 464s, 664s & 6128s in one way or another. Let us hope the decision is a wise one.

ACU for Sale

It was revealed last month that Amstrad Computer User was up for sale after the company who print it ran into severe difficulties and is now working under receivership. The kindest thing to do may be to let ACU die as its poor content is reflected in its dwindling readership. Whether any other company is willing to risk trying to restore ACU to some of its former glory is unknown. Though it would be ironic if the 'official' Amstrad magazine was about to disappear, just as Amstrad prepare to launch a new range of CPC computers.

ADVENTURES improved

Tony Kingsmill, author of 'Lords of Magic' and 'Island of Chaos' (reviewed in Issue Four), has now managed to iron out a couple of bugs in these programs which will allow them to run on both the CPC 464 and 6128. He has also removed a number of spelling errors from his programs and this will have improved their quality considerably. Print-Out has also learnt that a new adventure is on the way. Called 'Revenge of Chaos' it is the sequel to 'Island of Chaos' and in the game Baktron has returned and once more you will have to do battle with him. It is not expected to be released for a couple of months but we'll bring you more details when they are available.

We can also inform you that Tony is preparing a PD Library for the CPC on disc at present and it will hopefully be operational sometime soon - we'll give more information when the library is open.

PD on DISC

It seems that CPC owners have never had it so good!!! More and more small companies are setting up, especially in the PD field. Last issue we brought you news of CPD (a cassette public domain library) run by Alan Scully. We can tell you that CPD has expanded to also cover disc software and has changed its name to SCULL PD - of course the cassette side of things has not been forgotten. In this issue, we include an article on the new services & we can vouch for their excellence - if you're into PD it's well worth a look.

We are always looking for new writers, programmers or artists to help with PRINT-OUT. We are especially interested in articles on topics such as BASIC programming, hardware projects, business software, other languages, uses of CPCs and machine code programming.

If you are interested in helping with any of these things or anything else then please write to this address :-

PRINT-OUT, 8 MAZE GREEN ROAD, BISHOP'S STORTFORD, HERTFORDSHIRE CM23 2PJ.

LOCO by Anthony Millbourne

This program was written by Anthony Millbourne and it was his entry into the competition which we ran in Issue Two of the magazine; it won him second place. The object of the game is to try and stop the train coming off the tracks & to do this you have to move pieces of track around the screen so that there aren't any dead ends or breaks in the track. Unfortunately there's only one free place on the screen (the rest of the screen is filled up with random bits of track) & the only way to move a piece of track, is to position your cursor over a bit of track next to the hole and press fire. The piece of track will then be moved to the place where the gap was. The train moves around the track (shown by a piece of a different colour) and your game ends when it becomes derailed. Good Luck!!

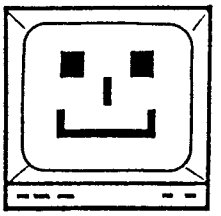
```
[3A] 10 MODE 1:INK 1,24:INK 3,6:INK 2,18:INK 0,1:PEN 1
       :rit$="M":lft$="N":up$="A":dwn$="Z":fir$=" "
[AD] 20 SYMBOL AFTER 220:FOR n=1 TO 32
[8A] 30 READ a,b,c,d,e,f,g,h
[42] 40 SYMBOL 223+n,a,b,c,d,e,f,g,h:NEXT
[C7] 50 DIM block(20,10):yx=10:yy=5:sx=2:sy=2
[99] 60 CLS:LOCATE 13,2:PRINT "WHICH LEVEL ?"
[98] 70 l=8:LOCATE 1,8:PRINT "          Easy   1
           Medium 2             Hard   3"
[83] 80 PRINT "                                                DEFINE
       KEYS"
[2D] 90 LOCATE 22,1:PRINT "(#)"
[C8] 100 k$=UPPER$(INKEY$):IF k$="" THEN GOTO 100
[92] 110 IF JOY(0) AND 1 AND l>8 OR k$=up$ AND l>8 THEN LOCATE 22,1:PR
       INT " ":l=l-2
[D8] 120 IF JOY(0) AND 2 AND l<14 OR k$=dwn$ AND l<14 THEN LOCATE 22,1
       :PRINT "   ":l=l+2
[BC] 130 IF JOY(0) AND 16 OR JOY(0) AND 32 OR k$=fir$ THEN GOTO 150
[D4] 140 GOTO 90
[BB] 150 IF l=8 THEN sp=40
[D5] 160 IF l=10 THEN sp=30
[DB] 170 IF l=12 THEN sp=20
[96] 180 IF l=14 THEN GOTO 1250
[2B] 190 k$="":CLS:yx=10:yy=5:sx=2:sy=2
[D9] 200 FOR n=1 TO 10:FOR a=1 TO 20
[7C] 210 block(a,n)=CINT(RND*6)+1
[1E] 220 IF a=1 THEN block(a,n)=6
[6D] 230 IF a=1 AND n=1 THEN block(a,n)=1
[D5] 240 LOCATE a*2-1,n*2-1:PRINT CHR$(block(a,n)*4+224)+CHR$(block(a,
       n)*4+225):LOCATE a*2-1,n*2:PRINT CHR$(block(a,n)*4+226)+CHR$(
       block(a,n)*4+227)
[AE] 250 NEXT a,n
[6B] 260 PEN 2:LOCATE 1,21:PRINT STRING$(40,207)
[4A] 270 LOCATE 1,22:PRINT CHR$(207):LOCATE 40,22:PRINT CHR$(207)
[0C] 280 LOCATE 1,23:PRINT STRING$(40,207):PEN 1
[49] 290 tx=1:ty=10:tbx=1:tby=11:t=19:sc=10
[B7] 300 block(sx,sy)=0:LOCATE sx*2-1,sy*2-1:PRINT "  ":LOCATE sx*2-1,
       sy*2:PRINT "  "
[98] 310 PEN 3:LOCATE 3,22:PRINT              "*LOCO*      SCORE ";sc:
       LOCATE 33,22:PRINT "*LOCO*"
[A4] 320 IF sc>1000 THEN GOTO 1140
[F2] 330 GOSUB 780
[63] 340 PEN 3:LOCATE yx*2-1,yy*2-1:PRINT CHR$(block(yx,yy)*4+224)+CHR
       $(block(yx,yy)*4+225):LOCATE yx*2-1,yy*2:PRINT CHR$(block(yx,
       yy)*4+226)+CHR$(block(yx,yy)*4+227)
[53] 350 PEN 1:LOCATE yx*2-1,yy*2-1:PRINT CHR$(block(yx,yy)*4+224)+CHR
       $(block(yx,yy)*4+225):LOCATE yx*2-1,yy*2:PRINT CHR$(block(yx,
       yy)*4+226)+CHR$(block(yx,yy)*4+227)
[87] 360 IF yx=tx AND yy=ty THEN GOTO 1170
[2C] 370 k$=UPPER$(INKEY$):IF k$="" THEN GOTO 300
[04] 380 IF JOY(0) AND 1 AND yy>1 OR k$=up$ AND yy>1 THEN yy=yy-1
[9A] 390 IF JOY(0) AND 2 AND yy<10 OR k$=dwn$ AND yy<10 THEN yy=yy+1
[64] 400 IF JOY(0) AND 8 AND yx<20 OR k$=rit$ AND yx<20 THEN yx=yx+1
[7D] 410 IF JOY(0) AND 4 AND yx>1 OR k$=lft$ AND yx>1 THEN yx=yx-1
[F8] 420 IF JOY(0) AND 16 OR JOY(0) AND 32 OR k$=fir$ THEN GOSUB 760
[F9] 430 GOTO 300
[6E] 440 DATA &00,&00,&00,&00,&00,&00,&00,&00
[70] 450 DATA &00,&00,&00,&00,&00,&00,&00,&00
[72] 460 DATA &00,&00,&00,&00,&00,&00,&00,&00
[74] 470 DATA &00,&00,&00,&00,&00,&00,&00,&00
[06] 480 DATA &00,&00,&00,&00,&00,&03,&04,&04
[36] 490 DATA &00,&00,&00,&1F,&E0,&00,&00,&00
[82] 500 DATA &08,&08,&08,&10,&10,&10,&10,&10
[9D] 510 DATA &00,&00,&00,&00,&03,&04,&08,&08
[31] 520 DATA &00,&00,&00,&E0,&1F,&00,&00,&00
[2E] 530 DATA &00,&00,&00,&00,&00,&C0,&20,&20
[58] 540 DATA &00,&00,&00,&00,&C0,&20,&10,&10
[9E] 550 DATA &10,&10,&10,&08,&08,&08,&08,&08
[32] 560 DATA &10,&10,&10,&10,&10,&08,&08,&08
[ED] 570 DATA &08,&08,&04,&03,&00,&00,&00,&00
[23] 580 DATA &04,&04,&03,&00,&00,&00,&00,&00
[3F] 590 DATA &00,&00,&00,&E0,&1F,&00,&00,&00
[3D] 600 DATA &10,&10,&20,&C0,&00,&00,&00,&00
[F3] 610 DATA &08,&08,&08,&08,&08,&10,&10,&10
[A6] 620 DATA &00,&00,&00,&07,&F8,&00,&00,&00
[6B] 630 DATA &20,&20,&C0,&00,&00,&00,&00,&00
[5A] 640 DATA &00,&00,&00,&FF,&00,&00,&00,&00
[5C] 650 DATA &00,&00,&00,&FF,&00,&00,&00,&00
[0E] 660 DATA &00,&00,&00,&00,&FF,&00,&00,&00
[10] 670 DATA &00,&00,&00,&00,&FF,&00,&00,&00
[40] 680 DATA &10,&10,&10,&10,&10,&10,&10,&10
[FA] 690 DATA &08,&08,&08,&08,&08,&08,&08,&08
[31] 700 DATA &10,&10,&10,&10,&10,&10,&10,&10
[EB] 710 DATA &08,&08,&08,&08,&08,&08,&08,&08
[74] 720 DATA &10,&10,&10,&F0,&00,&00,&00,&00
[FF] 730 DATA &08,&08,&08,&0F,&00,&00,&00,&00
```

```
[8C] 740 DATA &00,&00,&00,&00,&F0,&10,&10,&10
[BB] 750 DATA &00,&00,&00,&00,&0F,&08,&08,&08
[85] 760 IF yy=sy+1 AND yx=sx OR yy=sy-1 AND yx=sx OR yx=sx+1 AND yy=
     sy OR yx=sx-1 AND yy=sy THEN GOTO 770 ELSE RETURN
[EC] 770 block(sx,sy)=block(yx,yy):LOCATE sx*2-1,sy*2-1:PRINT CHR$(blo
     ck(sx,sy)*4+224)+CHR$(block(sx,sy)*4+225):LOCATE sx*2-1,sy*2:
     PRINT CHR$(block(sx,sy)*4+226)+CHR$(block(sx,sy)*4+227):block
     (yx,yy)=0:sx=yx:sy=yy:RETURN
[34] 780 t=t+1:IF t>sp THEN t=0 ELSE RETURN
[FB] 790 PEN 1:LOCATE tx*2-1,ty*2-1:PRINT CHR$(block(tx,ty)*4+224)+CHR
     $(block(tx,ty)*4+225):LOCATE tx*2-1,ty*2:PRINT CHR$(block(tx,
     ty)*4+226)+CHR$(block(tx,ty)*4+227)
[8F] 800 IF block(tx,ty)=1 THEN GOTO 880
[B7] 810 IF block(tx,ty)=2 THEN GOTO 900
[12] 820 IF block(tx,ty)=3 THEN GOTO 920
[6D] 830 IF block(tx,ty)=4 THEN GOTO 940
[C8] 840 IF block(tx,ty)=5 THEN GOTO 960
[23] 850 IF block(tx,ty)=6 THEN GOTO 980
[02] 860 IF block(tx,ty)=7 THEN GOTO 1000
[FF] 870 END
[4C] 880 IF tbx=tx+1 THEN GOTO 1110 ELSE GOTO 1050
[03] 890 END
[09] 900 IF tbx=tx-1 THEN GOTO 1110 ELSE GOTO 1090
[F4] 910 END
[D3] 920 IF tbx=tx+1 THEN GOTO 1070 ELSE GOTO 1050
[F8] 930 END
[A3] 940 IF tbx=tx-1 THEN GOTO 1070 ELSE GOTO 1090
[FC] 950 END
[15] 960 IF tbx=tx+1 THEN GOTO 1090 ELSE GOTO 1050
[00] 970 END
[71] 980 IF tby=ty+1 THEN GOTO 1070 ELSE GOTO 1110
[04] 990 END
[4D] 1000 IF tbx=tx+1 THEN GOTO 1090
[F6] 1010 IF tbx=tx-1 THEN GOTO 1050
[30] 1020 IF tby=ty+1 THEN GOTO 1070
[BA] 1030 IF tby=ty-1 THEN GOTO 1110
[2D] 1040 END
[16] 1050 tbx=tx:tby=ty:tx=tx+1:IF tx>20 OR block(tx,ty)=1 OR block(tx
     ,ty)=3 OR block(tx,ty)=6 OR block(tx,ty)=0 THEN GOTO 1170
[3E] 1060 GOTO 1120
[B6] 1070 tbx=tx:tby=ty:ty=ty-1:IF ty<1 OR block(tx,ty)=3 OR block(tx,
     ty)=4 OR block(tx,ty)=5 OR block(tx,ty)=0 THEN GOTO 1170
[44] 1080 GOTO 1120
[63] 1090 tbx=tx:tby=ty:tx=tx-1:IF tx<1 OR block(tx,ty)=2 OR block(tx,
     ty)=4 OR block(tx,ty)=6 OR block(tx,ty)=0 THEN GOTO 1170
[2E] 1100 GOTO 1120
[97] 1110 tbx=tx:tby=ty:ty=ty+1:IF ty>10 OR block(tx,ty)=1 OR block(tx
     ,ty)=2 OR block(tx,ty)=5 OR block(tx,ty)=0 THEN GOTO 1170
[05] 1120 PEN 3:LOCATE tx*2-1,ty*2-1:PRINT CHR$(block(tx,ty)*4+224)+CH
     R$(block(tx,ty)*4+225):LOCATE tx*2-1,ty*2:PRINT CHR$(block(t
     x,ty)*4+226)+CHR$(block(tx,ty)*4+227)
[9B] 1130 sc=sc+(50-sp):RETURN
[69] 1140 PEN 2:CLS:LOCATE 16,10:PRINT "NEXT LEVEL !"
[21] 1150 IF INKEY$="" THEN GOTO 1150
[57] 1160 PEN 1:sp=sp-5:GOTO 190
```

Linechecker

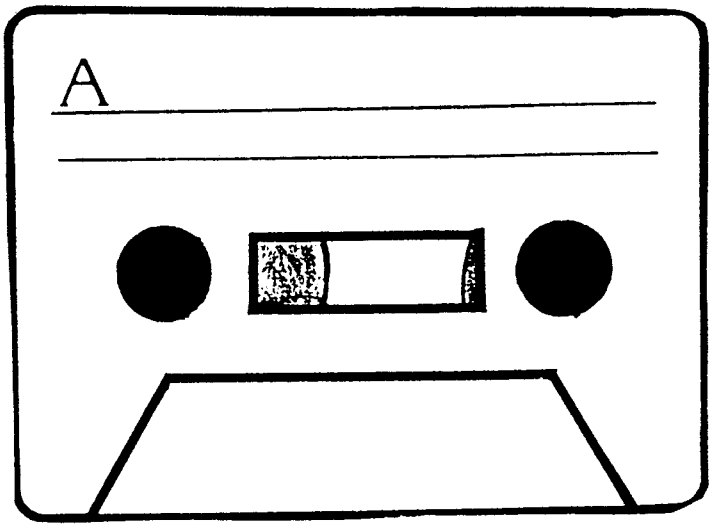
A PROGRAM TYPING AID

All programs in Print-Out have Linecheck codes which are enclosed in brackets at the start of a line. Don't enter them in as they're designed to be used with Linechecker to eliminate errors when typing in programs which appear in this magazine. Please note, all programs will run whether Linechecker is being used or not. For information on how to use Linechecker, please see Issue Three.



```
[FE] 1170 CLS:PEN 2:LOCATE 8,4
[C0] 1180 PRINT "        #####  #  #   # #####
            #   # # # ## ## #                      #     ###
          # # # #####   # ### # # #   # #
           #   # # # ## ###### # # #   # #####";
[DE] 1190 PRINT:PRINT:PRINT "        ##### # # ##### ####
                #   # # # #     #   #                #   #
          # # ######   # #   # # # #     ####
            #   # ## #   #   #                  #####  #  ###
          ## #   #"
[97] 1200 IF INKEY$="" THEN GOTO 1200
[89] 1210 PEN 1:GOTO 60
[0D] 1250 CLS:LOCATE 1,2:PRINT "             DEFINE KEYS"
[0B] 1260 LOCATE 1,8:INPUT "RIGHT ";rit$
[78] 1270 LOCATE 1,10:INPUT "LEFT  ";lft$
[45] 1280 LOCATE 1,12:INPUT "UP    ";up$
[85] 1290 LOCATE 1,14:INPUT "DOWN  ";dwn$
[BE] 1295 LOCATE 1,16:INPUT "FIRE  ";fir$
[0E] 1300 rit$=UPPER$(rit$):lft$=UPPER$(lft$):up$=UPPER$(up$):
     dwn$=UPPER$(dwn$):fir$=UPPER$(fir$):GOTO 60
```

Offers



Please make all cheques payable to Print-Out but any postal orders should be made out to T J Defoe as this saves the Post Office a great deal of time and effort. Unless it cannot be avoided, it is advisable not to send cash through the post. All orders should be sent to :- PRINT-OUT, Special Offers. 8 Maze Green Road. Bishop's Stortford. Hertfordshire CM23 2PJ.

Issue 6

If you wish to order a copy of Issue Six in advance, you may do so by sending a cheque / postal order for £1.10 (or 70p + an A4 SAE with a 28p stamp) to the usual address. We hope to have it published by about the 30th July & as soon as it is printed it will be forwarded to you.

Program tapes and discs

We now supply both program tapes and discs for ALL issues and the prices given below also include a booklet to explain how the programs work plus postage and packing. Tapes & discs are available for Issues One, Two, Three, Four & Five. The cost for a program tape is as follows :-

a) A blank tape (at least 15 minutes) and 50p (p+p)
or b) £1.00 (which also includes the price of a tape)

The cost for a program disc is :-

a) A blank formatted disc and 50p (p+p)
or b) £3.00 (which also includes the cost of a MAXELL/AMSOFT disc) *

* When ordering using this particular method. please allow about 14 days for delivery as we must rely on outside suppliers for the discs.

Back issues

We still have some copies of Issues 1, 2, 3 and 4 available and the price is £1.10 which includes postage and packing. Alternatively, you can order both a back issue and its corresponding tape or disc by sending :-

a) £1.75 - includes the tape, the required issue and postage and packing
b) £3.75 - includes the disc (genuine MAXELL/AMSOFT disc) & the required issue and postage and packing.

Send to: Print-Out, 8 Maze Green Road,
Bishop's Stortford, Herts.

NAME (block capitals please) .....
ADDRESS .....
.....
.....
POSTCODE .....

Please send me the following items :-

DESCRIPTION	ISSUE NUMBER	QUANTITY	PRICE EACH	PRICE
			TOTAL PRICE	

I enclose a cheque/postal order/cash* to the value of £..... Please make all cheques payable to PRINT-OUT and make postal orders out to Thomas Defoe. Thank you. (*delete as applicable)

Subscription

If you are interested in having a subscription to Print-Out you will be glad to know that full details concerning subscriptions are printed on the next page. We are running two types of subscription - half-yearly (three issues) and also yearly (six issues) at the prices of £3.30 and £6.60 respectively.

Advertising

You can place a small ad of upto 40 words (including your name and address) in this section of the magazine free of charge. If you wish to place a larger advertisement in the magazine, please write to us for a full list of advertising rates.

Subscription Offer

Due to the many enquiries that we have received concerning subscriptions we have now introduced a subscription service and it will be operating from Issue Five. There are two forms of subscription :-

a) Three issues - approxiamately half a year

b) Six issues - approxiamately a full year

Although we do try and produce one magazine every two months this is not always possible due to other outside engagements and therefore exact release dates are not given in the magazine. Because of this, we are unable to guarantee that six issues will be produced in a year, or three issues in half a year. However, for a year's subscription you will be sent six issues no matter when they are published, and the same applies to a half-yearly subscription. If we stop producing the magazine, we promise to refund the cost of all unmailed issues.

The prices for subscriptions to Print-Out are as follows :-

NO OF ISSUES	UNITED KINGDOM	EUROPE	REST OF THE WORLD
SINGLE	£1.10	£1.50	£2.00 / £2.50*
THREE ISSUES	£3.30	£4.50	£6.00 / £7.50*
SIX ISSUES	£6.60	£9.00	£12.00 / £15.00*

* The first price quoted is for 'Printed Paper Rate' but this does not have the same level of security as a normal letter or parcel. The second price mentioned is for normal rate and is sent in the same way as an ordinary letter or parcel. For the United Kingdom there is an alternative price for ordering only a single issue and this is:- 70p + a large A4 SAE (with 28p stamp). We try to despatch all orders within a week of receiving them & all items are sent by SECOND CLASS post, unless the extra money or stamps are sent with your order.

Subscriptions

Send to: Print-Out, 8 Maze Green Road,
Bishop's Stortford, Herts.

NAME (block capitals please) .....

ADDRESS .....,

.....,

.....

POSTCODE .....

Please send me the next three/six* issues of Print-Out as soon as they are published. I enclose a cheque/postal order* to the value of £..... and I wish my subscription to start from Issue ..... (* delete as applicable) Please make CHEQUES payable to PRINT-OUT and make postal orders payable to Thomas Defoe (as this saves time and effort for the Post Office !!!).